VLDB journal manuscript No.
(will be inserted by the editor)

Qualitative Effects of Knowledge Rules and User Feedback in Probabilistic Data Integration

Maurice van Keulen · **Ander de Keijzer**

11 June 2009

Abstract In data integration efforts, portal development in particular, much development time is devoted to entity resolution. Often advanced similarity measurement techniques are used to remove semantic duplicates or solve other semantic conflicts. It proves impossible, however, to automatically get rid of all semantic problems. An often-used rule of thumb states that about 90% of the development effort is devoted to semi-automatically resolving the remaining 10% hard cases. In an attempt to significantly decrease human effort at data integration time, we have proposed an approach that strives for a 'good enough' initial integration which stores any remaining semantic uncertainty and conflicts in a probabilistic database. The remaining cases are to be resolved with user feedback during query time. The main contribution of this paper is an experimental investigation of the effects and sensitivity of rule definition, threshold tuning, and user feedback on the integration quality. We claim that our approach indeed reduces development effort — and not merely shifts the effort — by showing that setting rough safe thresholds and defining only a few rules suffices to produce a 'good enough' initial integration that can be meaningfully used, and that user feedback is effective in gradually improving the integration quality.

1 Introduction

Data integration is a challenging problem in many application areas as it usually requires manual resolution of semantic issues like schema heterogeneity, data overlap, and data inconsistency. In this paper we focus on data overlap as a major source of semantic uncertainty and conflicts, hence for the need for human involvement. Data overlap occurs when data sources contain data about the same real world objects (rwos). For example, when developing a portal such as DBlife [DS$^+$07], one strives for gathering as much information as possible related to a specific set of rwos i.e. people, from various sources [DSC$^+$07]. It is, however, mostly not possible to determine with certainty whether or not data items refer to the same rwo or not. This problem is usually referred to as *entity resolution*.

Advanced similarity measurement techniques can be used to remove semantic duplicates and other semantic conflicts, but it proves impossible to automatically get rid of all semantic problems. An often-used rule of thumb states that about 90% of the development effort is devoted to solving the remaining 10% hard cases, because human knowledge is required to ultimately decide if two data items refer to the same rwo and, if so, how to resolve conflicts between the two. Note that strictly speaking, even for humans making an absolute decision may be extremely labor-intensive. Figure 1 illustrates this: although the substring "beth" is the only similarity hinting at the possibility that these two data items refer to the same rwo, it may very well be that this is the case, namely a woman who recently got married and moved in with her husband; only really contacting this person may ultimately resolve the issue.

Data source 1

name:	Elisabeth Johnson
address:	Wall street 12
phone:	555-823 5430

Data source 2

name:	Beth Clark
address:	Robertson Ave 2
phone:	576-234 8751

Fig. 1 Example instances of two data sources with address cards

Most data integration approaches require resolution of semantic uncertainty and conflicts before the integrated data can be meaningfully used [DH05]. We believe, however, that

Maurice van Keulen
Faculty of EEMCS, University of Twente, P.O. Box 217, 7500AE, Enschede, The Netherlands. E-mail: m.vankeulen@utwente.nl

Ander de Keijzer
Institute of Technical Medicine, Faculty of Science and Technology, University of Twente, P.O. Box 217, 7500AE, Enschede, The Netherlands. E-mail: a.dekeijzer@utwente.nl.

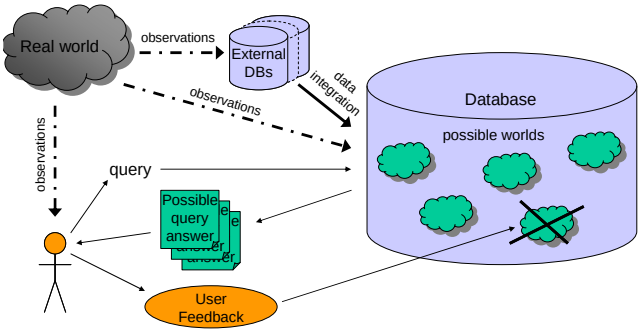


Fig. 2 Information Cycle

data integration can be made into less of a development obstacle by removing this restriction striving for less perfect, but near-automatic integration, i.e., "good is good enough" data integration. The idea behind our probabilistic data integration approach is to postpone resolution of the remaining 10% semantic uncertainty to a moment more natural to human involvement namely during querying. We strive for an initial integration which stores any remaining semantic uncertainty in a probabilistic database. This allows the integrated data to already be used after 10% of the development effort. This is not only a development benefit. As argued by [Orr98] "the only way to truly improve data quality is to increase the use of that data". Being able to properly handle uncertainty in data can provide for near-automatic data integration, hence an earlier chance for getting the user in the loop (our use of user feedback is a form of real-world feedback control system claimed of vital importance in [Orr98]).

A schematic overview of our approach is given in Figure 2 [KKA05, dKvK08]. We view a database as a representation of information about the real world based on observations. In this view, data integration is a means to combine independent observations from different data sources. Since we focus on data overlap, we assume that the schemas of the data sources are already aligned. The DBMS becomes uncertain about the state of the real world when observations conflict or cannot be traced back to RWOs with certainty. We have chosen a representation of uncertain data that compactly represents in one XML tree all possible states the real world can be in, the *possible worlds*. Posing queries to an uncertain database means that an application may receive several possible answers. In many application areas, this suffices if those answers can be properly ranked according to likelihood. A user interacting with an application can provide feedback on the correctness or plausibility of these answers. This feedback can be traced back to possible worlds, hence be used to remove *impossible* worlds from the representation in the database. This incrementally improves the quality of the integrated data.

The general architecture of the system is depicted in Figure 3. A compact representation of the possible worlds

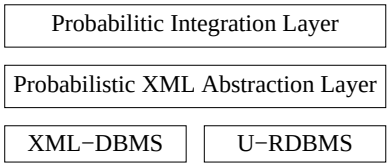


Fig. 3 System architecture

combines the notions of an XML database with an uncertain database. Therefore, it can be implemented both on top of an XML DBMS by adding functionality of handling uncertainty, or on top of an uncertain relational DBMS (U-RDBMS) by adding functionality of handling XML. This functionality is provided by the probabilistic XML abstraction layer. We currently only fully support the former; the latter is under development. The probabilistic data integration functionality is implemented on top of this. We focus in this paper on the probabilistic integration layer.

An automatic process can in theory never make an absolute decision for entity resolution. However unlikely, there are always situations imaginable where data items that completely differ still refer to the same RWO (as in Figure 1) or where similar data items refer to different RWOs. But, if a probabilistic integration system would consider and store all theoretically possible alternatives, the integrated data set would explode in size. Therefore, a human is still needed to reduce the possibilities by specifying knowledge rules that make absolute decisions. An example of such a rule could be: "if, according to some distance measure, address cards are further away than some threshold, assume that they do not refer to the same RWO". We observed that simple rules ruling out non-sensical alternatives often suffice to reduce the amount of uncertainty to a manageable size [dKvKL06].

To prove, however, that our approach indeed reduces development effort significantly — and not merely shifts the effort to rule definition and threshold tuning — we show that setting rough thresholds and defining only a few rules suffices to produce a 'good enough' initial integration that can be meaningfully used and can be effectively improved using user feedback. 'Good enough' here refers to the quality of the integration result. We define the quality of an integrated data set by means of the quality of the possible answers to queries. Ruling out *incorrect* possibilities not only results in a reduction in size of the integrated data set, it also results in a better *quality* integration, because incorrect possibilities lead to incorrectness in the answers. When rules make absolute decisions, they may, however, make a wrong decision (e.g., the given rule would make the wrong decision for Figure 1). Ruling out *correct* possibilities in this way leads to a lower quality integration result. Therefore, there is a trade-off between a developer trying to specify stricter rules to reduce the amount of uncertainty and increase integration quality, and the likelihood that his rules make wrong deci-

sions, which decreases integration quality. Observe that current data scrubbing and entity resolution techniques make an absolute decision at integration time for all data items, hence are likely to make such mistakes and therefore do not produce the best quality integration result.

The effects of rules, thresholds and user feedback on integration quality is far from trivial, because, for example, a particular case of semantic uncertainty may affect one query and not another. Moreover, there often exist dependencies between semantic conflicts. Upfront, it is also not evident how big the impact of additional uncertainty or user feedback is on the integration quality. In other words, how rule definition, threshold tuning, and user feedback precisely affect integration quality needs to be investigated experimentally with real-life data. Such an investigation is the focus of this paper: we present an experimental investigation of the effects and sensitivity of rule definition, threshold tuning, and user feedback on the integration quality.

Albeit an intuitively attractive notion, 'integration quality' is rather vague. Therefore, we have defined information retrieval-like query answer quality measures based on precision and recall. The statistical notion of *expected value* when applied to precision and recall naturally takes into account the probability with which a system claims query answers to be true.[1] The quality of a correct answer is higher if the system dares to claim that it is correct with a higher probability. Analogously, incorrect answers with a high probability are worse than incorrect answers with a low probability.

The trade-off of a developer trying to specify stricter rules and thresholds to reduce the amount of uncertainty and increase integration quality, and the likelihood that his rules make wrong decisions which decreases integration quality, can be more precisely defined in this way. We forsee that as long as a developer's rules and thresholds do not make wrong decisions, precision goes up and recall remains the same with stricter rules and thresholds. When they become too strict and start to make wrong decisions, precision and recall go down. Our hypothesis, however, is that

1. Precision and recall are not very sensitive to safe thresholds, hence developers can save much effort on threshold tuning by taking rough but safe thresholds,
2. just a few rules suffice to achieve acceptable initial integration quality, hence not much effort is required for rule definition, and
3. user feedback is effective in quickly improving integration quality.

[1] In [dKvK07a], we also proposed adapted notions of precision and recall. The latter coincided with the expected value of recall, but the former does not. We have chosen to use the expected value of precision in this paper instead of the adapted precision measure of [dKvK07a].

Fig. 4 Illustration of differences between our data sets (important ones in bold).

1.1 Contributions

- An overview of our probabilistic XML data integration approach containing several (small) improvements on our earlier published work [KKA05, dKvKL06, dKvK07a, dKvK07b, dKvK08],
- an experimental investigation of the sensitivity of rule definition, threshold tuning, and user feedback on integration quality, and
- based on the insights we provide experimental evidence that our probabilistic integration approach is indeed effective in significantly reducing development effort.

1.2 Running example

As a running example and set-up for our experiments, we selected a typical portal application that requires integration

of data sources on the Internet. The purpose of the portal is to collect information on movies that are about to be aired on TV. It uses an Internet TV guide[2] for finding out which movies are about to be aired as well as other information the website provides for these movies. The portal furthermore *enriches* this information with data from IMDB[3]. With *enrichment* we mean adding information from another source about a certain set of entities. In our experiments, we enrich our information about movies with information about genres, actors, directors, locations, keywords, and plots.

Figure 4 shows the data about the movie 'The Namesake' from both data sources as an illustration of the conflicts and entity resolution problems the portal faces. One entity resolution problem is that it cannot determine with certainty which movie of the TV guide corresponds with which movie in IMDB. We have observed typos and different naming conventions in the titles (e.g., 'The Namesake' vs. 'Namesake, The'). Furthermore, both sources have data on actors and their roles in the movie. This poses a second entity resolution problem: the application cannot determine with certainty which actors and roles correspond. There are many typos and differences in naming conventions in actor names and role descriptions (e.g., 'Glenne Headley' with role 'Lydia' vs. 'Headly, Glenne' with role 'Lydia Ratliff'). The role of Ashoke's Father poses a major semantic conflict: 'Tamal Roy Choudhury' vs. 'Tamal Sengupta' for which even a human would doubt that it is the same actor.

1.3 Overview

The paper is organized as follows. Section 2 describes related research. We then introduce our probabilistic integration approach in Section 3. Since measurement of the usually hard to grasp concepts of uncertainty and quality plays a vital role in this research, we devote an entire section to this topic (Section 4). We then present the experiments concerning rule definition and threshold tuning in Section 5. The user feedback experiments are presented in Section 6. Finally, we present conclusions and future work in Section 7.

Note that since this paper is only concerned with *qualitative* effects, we focus on the probabilistic integration layer of Figure 3. We only superficially describe the algorithms for querying and user feedback and leave out experiments dealing with execution performance and scalability. Any algorithm in the probabilistic XML abstraction layer will need to adhere to the same semantics, hence algorithmic differences have no influence on the qualitative effects we study here and are beyond the scope of the paper. Although we present the semantics of all notions in terms of possible worlds, in reality our algorithms perform queries and feedback directly on the compact representation thus avoiding enumeration of possible worlds.

[2] `www.tvguide.com`
[3] `www.imdb.com`

2 Related Research

Probabilistic databases. Several models for uncertain data have been proposed over the years. Initial efforts focused on relational data [BGMP90]. Current efforts made in the relational setting remain strong [LLRS97, BSHW06, BDM+05, CSP05, AKO07]. Two methods to associate confidences with data are commonly used. The first method associates the confidence scores with individual attributes (e.g., [BGMP90]), whereas the second method associates these confidence scores with entire tuples (e.g., [BSHW06]).

Semistructured data, and in particular XML, has also been used as a data model for uncertain data [HGS03, AS06, KKA05]. There are two basic strategies. The first strategy is *event based* uncertainty, where choices for particular alternatives are based on specified events [AS06, HGS03]. Nodes are annotated with event expressions which validate or invalidate the node and its subtree according to combinations of occurrences of events. Events are assumed to be independent of each other. One particular combination of events represents a possible world, hence all possible worlds are obtained by enumerating all possible combinations.

The other strategy for semistructured models is *choice point based* uncertainty [KKA05]. At specific points in the tree a node representing a choice between subtrees is inserted. Choosing one child node, and as a result an entire subtree, invalidates the other child nodes. As with the event based strategy, one particular possible world can be selected by making a choice for each choice point. The model presented in this paper is based on the choice point strategy.

Data integration. The amount of work on information integration is enormous. The topic has been studied for several decades already and will remain a research question for many more to come. This is due to the semantics captured in the schema and data. Since semantics is impossible to handle by a machine, human involvement will always be necessary to make final decisions on semantic issues. A first and already well-studied challenge in information integration is schema matching. The result of this process is a mapping between data sources relating not only element types, but also providing mapping functions that indicate how the *data* should be transformed from one source to the other.

A recent overview of schema integration is given in [DH05]. The Learning Source Descriptions (LSD) project [DDH01, DDH03] from the same authors is widely recognized as a big step forward in the schema integration field. In this project, machine learning techniques are applied to

effectively use data *instances* for schema matching. Furthermore, it employs a multi-strategy approach where clues obtained from several base learners are combined into a joint similarity estimate by a meta learner.

Although the schema integration phase is an important and difficult part of the entire integration process, it is not the focus of this paper. In this paper, we assume that schema integration has already taken place and we focus on the integration issues in the instance data. The one work we found that also uses a probabilistic XML representation in an attempt to integrate XML documents is [HL06]. Others that argued for explicitly handling the uncertainty involved in data integration [MM07, DHY07, SDH08, Gal08] focus on uncertainty pertaining to the mappings produced by a schema matcher. Nevertheless, this also creates uncertainty about existence of tuples in the integrated database.

Entity Resolution. The data integration problem we focus on is entity resolution or record linkage. Best fitting to our focus on resolving entities in XML data is [MSC06] which presents an XML tree distance measure that explicitly takes into account the structure but not the order in the tree. In [BGMM+09], a generic approach to entity resolution is presented. The method is generic in the sense that both comparing and merging of records are viewed as black-boxes. In an earlier paper [MBGM06], the generic approach is used to also manage confidences along with the data. [DSC+07] describes Cimple, an approach for extracting entities from unstructured (textual) sources. An algebra with extraction operators is presented which allows to combine evidences from different sources. For example, for extracting researchers mentioned as PC-members in conference notifications for the DBlife-portal [DS+07], the researchers are disambiguated by checking in DBLP whether or not two researchers have ever been co-authors, which is a highly domain-specific but accurate technique for entity resolution for researcher entities. Closely related is the topic of *entity search* [CYC07, SH08], search engines not geared towards finding web pages, but entities such as persons, experts or telephone numbers.

Note that in our work, we do not focus on the entity resolution problem itself, but on how to properly handle the inherent ambiguous decisions any entity resolution technique makes. By explicitly not using the advanced entity resolution techniques described above, we show that our approach even works for simple entity resolvers that make highly ambiguous decisions and often make mistakes.

User feedback for data quality improvement. User feedback with the purpose to improve data quality in uncertain data has received little attention. Closest related work we found is [KO08] which presents an approach to evaluate queries on uncertain relational data given that certain conditions hold. This can be used for conditioning a database based on new evidence. In Section 3.5, we discuss how the approach could be used for applying user feedback in probabilistic XML.

Relevance feedback is a form of user feedback that is well-studied in information retrieval [BYRN99]. Its aim is not to improve data, but the contextualization of queries.

Consistent query answering in inconsistent databases is also a well-studied problem (see e.g., [FGM07, Wij06]). This problem acknowledges the fact that data in a database may be inconsistent, i.e., does not conform to certain integrity constraints. The solution is not sought in explictly modelling the conflicts as uncertainty, but by looking for query answers that do not depend on the inconsistencies, more exactly, that are independent of any minimal repair that resolves the conflict. In our approach, we would obtain consistent answers by selecting only answers that have a probability of 1.

Finally, there are attempts at automatically improving data quality. For example, [CCX08] presents an entropy-based quality measure (PWS-quality) as basis for data cleaning geared towards reducing ambiguousness in query results. Note that ambiguousness is not the same as correctness as there is no user in the loop (see also the discussion in Section 4.3). Most of these automatic efforts focus on improving accuracy of sensor data (e.g., [KD08]).

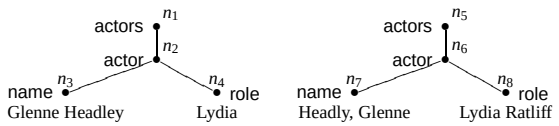
3 Probabilistic Data Integration

In this section, we present the probabilistic data integration approach used in IMPrECISE, our prototype used in the experiments. We explain the foundation in *possible worlds* theory, our representation for uncertain data, the integration algorithm itself and the rules used during integration.

3.1 Possible worlds

An ordinary database can be considered as a representation of (a part of) the real world. In an ideal system, this representation perfectly matches the real world. In many cases, however, this ideal cannot be reached. An uncertain database allows to store multiple possible representations for a real-world object (rwo) in case it is uncertain which representation is the correct one. In that sense, an uncertain database is a representation of possible worlds. Possible worlds are mutually exclusive, and as a consequence, at most one of the possible worlds is assumed to actually correctly represent the real world. Each possible world can have an associated probability indicating the confidence with which the database beliefs that the particular possible world correctly reflects the real world.

Definition 1 (possible worlds) Let $\mathscr{D}$ be the universe of database states. We vary D over $\mathscr{D}$. Then, $\mathscr{PD} = \mathbb{P}(\mathbb{R} \times \mathscr{D})$ is the universe of probabilistic database states where $\mathbb{P}$ denotes the power set constructor. We vary PD over $\mathscr{PD}$.



(a) Fragment TV guide (b) Fragment IMDB

Fig. 5 Example fragments from both sources (node identifiers n_i are included for reference in the explanation of algorithm 10)

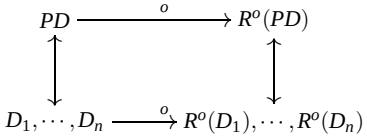


Fig. 8 Commutative diagram (o denotes an operation, R_o the result of o)

Example 1 To illustrate our approach for representing the uncertainty involved in data integration, let's suppose we like to integrate the example fragments of Figure 5. The example pertains to an actor list from our two sources each containing one actor, namely 'Glenne Headley'. As we have seen in Figure 4, data about this actor from both sources is conflicting. Suppose it is impossible at integration time to make an absolute decision whether both actor elements refer to the same rwo, i.e., the system estimates that they match with probability α. Based on these facts and assumptions, there are 5 possible worlds: In one world there are two actors (the case that they do not refer to the same rwo). Since there are two possibilities for the name and independently two possibilities for the role, there are four possible worlds for the case that they do refer to the same rwo. Let us furthermore assume that there is no evidence that either name or role text is correct, i.e., the probability for both is 0.5. See Figure 6 for an illustration of this set of possible worlds.

Note that we already used a bit of domain knowledge here, namely, that the elements actors, name, and role can only occur once under their parent elements. Without this DTD knowledge, there would be many more possible worlds.

3.2 Compact representation of possible worlds

To capture uncertainty in the XML data model, we introduce two new node kinds: probability nodes ($\bigtriangledown$) to represent choice points and possibility nodes ($\circ$) to represent the alternatives.[4] Child nodes of probability nodes are always possibility nodes. Each possibility node has an associated probability, which denotes the confidence of the database that the node and its subtree correctly reflects the real world. Sibling possibility nodes are mutually exclusive and their probabilities should add up to 1, hence probability nodes indicate *choices*. Child nodes of possibility nodes are regular XML nodes ($\bullet$). Child nodes of regular XML nodes can be either other XML nodes or probability nodes.

More formally, let $\mathscr{T}_{\text{fin}}$ be the set of *ordered finite trees* representing XML documents. We vary T over $\mathscr{T}_{\text{fin}}$. Let $PT = (T, \mathsf{kind}, \mathsf{prob})$ be a *probabilistic tree* where the kind function assigns kinds to nodes and the prob function assigns probabilities to possibility nodes. A probabilistic tree is *well-formed* iff it adheres to the aforementioned restrictions. A formal definition of a slightly stricter probabilistic XML data model is given in [KKA05].

The root node of the tree in Figure 6 can be seen as one big choice point enumerating all possibilities. We call this the *possible world representation*. We can, however, obtain a more compact representation of the set of possible worlds by *pushing down* the choice points. If we do this for our example we arrive at the tree in Figure 7. We call this the *compact representation*. An XML representation of this tree is what IMPrECISE stores as its (uncertain) database state. Observe that the individual aspects of uncertainty in the example, i.e., (i) whether or not both actor subtrees refer to the same real world actor, and if so (ii) which name is the correct name, and (iii) which role is the correct role, are now separated and local to the elements with which they are associated. For smaller examples, the reduction in size is minimal, but with a growing number of worlds and much overlap between worlds, the size benefit is much larger.

A probabilistic tree PT represents a set of possible worlds PD. We obtain the set of possible worlds PWS_{PT} by constructing trees for all combinations of local possibilities. Let $\mathrm{P}(T \mid PT)$ be the probability of a possible world T given the probabilistic database PT. Note that $PD = \{(\mathrm{P}(T \mid PT), T) \mid T \in \mathsf{PWS}_{PT}\}$. PT_1 and PT_2 are called *equivalent* iff $\mathsf{PWS}_{PT_1} = \mathsf{PWS}_{PT_2}$. The trees in Figures 6 and 7 are equivalent. The compact representation basically is the probabilistic tree with the least number of nodes that is equivalent with the possible world representation.

3.3 Querying possible worlds

Querying, as all operations on a probabilistic database, should adhere to possible world theory. The theory dictates that the semantics of an operation on an uncertain database is the same as the combination of the evaluation of the operation on each world independently. Since a possible world is an ordinary database state, it is clear what the semantics is of the evaluation of the operation on one particular world, hence we can deduce the semantics of operations working on the probabilistic database. The commutative diagram in Figure 8 illustrates this principle. To illustrate that the prin-

[4] In actual XML documents, we represent a probability node with an element "prob" and a possibility node with an element "poss".

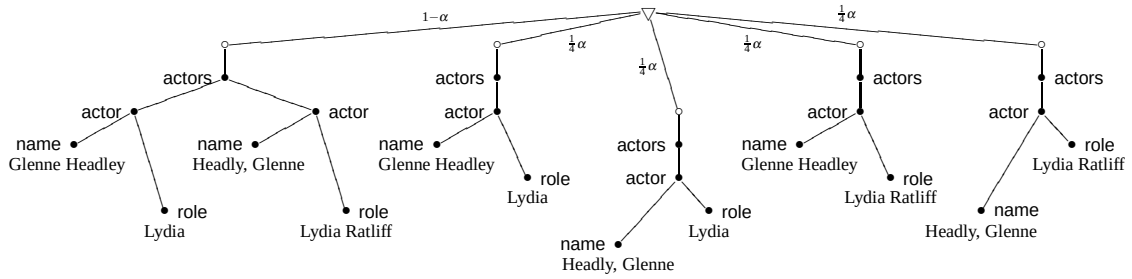


Fig. 6 Possible world representation

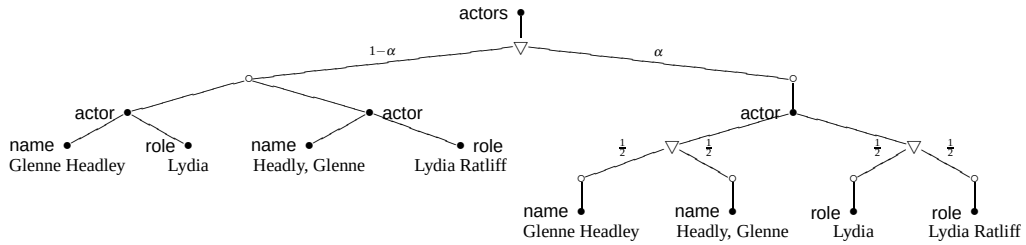


Fig. 7 Compact representation

ciple is data model and query language independent, we use D and PD instead of T and PT in this section.

Definition 2 (querying) Let $Q(D)$ be the semantics (i.e., result) of query Q on regular database D. Then, the set of all possible answers is defined by

$$\mathsf{Ans}_Q(PD) = \{Q(D) \,|\, (p,D) \in PD\}$$

Since the same answer may be produced by several possible worlds, the probability of an answer a is defined by

$$P(a \in Q(PD)) = \sum_{(p,D) \in PD \wedge a \in Q(D)} p$$

The semantics of query Q on a probabilistic database PD can now be defined as

$$[\![Q(PD)]\!] = \{(a,p) \,|\, a \in \mathsf{Ans}_Q(PD) \wedge p = P(a \in Q(PD))\}$$

For XPath or XQuery the result of an answer is a sequence. Possible answers usually only differ in the existence of one or more elements. For applications, it suffices and is more practicable to just know on a 'per-element' basis with which probability it occurs a certain number of times in the answer (its *multiplicity*).

Definition 3 (Per-element answer style) Let $\overline{\mathsf{Ans}}_Q(PD) = \{v^m \,|\, a \in \mathsf{Ans}_Q(PD) \wedge v^m \in a\}$ be the set of all possible occurrences of an element v where $v^m \in a$ denotes that v has multiplicity m, i.e., v occurs m times in a. The probability of v occurring m times in the result is defined as

$$P(v^m \in Q(PD)) = \sum_{(p,D) \in PD \wedge a \in Q(D) \wedge v^m \in a} p$$

'Per-element' *answer style* semantics can now be defined as

$$[\{Q(PD)\}] = \{(v^m, p) \,|\, v^m \in \overline{\mathsf{Ans}}_Q(PD) \wedge p = P(v^m \in Q(PD))\}$$

Example 2 Consider the movie document in Figure 6 and the query Q='Give me the role of the actor named Glenne Headley' expressed for example as the XPath query

```
//actor[name="Glenne Headley"]/role
```

Only 3 of the possible worlds will return a non-empty answer, two containing the role 'Lydia', one containing the role 'Lydia Ratliff'. Therefore, the result is

`Lydia`^{1}	$1-\alpha+\frac{1}{4}\alpha$
`Lydia`^{0}	$\frac{3}{4}\alpha$
`Lydia Ratliff`^{1}	$\frac{1}{4}\alpha$
`Lydia Ratliff`^{0}	$1-\alpha+\frac{3}{4}\alpha$

Note that the probabilities for each element v correctly add up to one.

3.4 Query algorithm

A naive implementation following possible worlds theory closely is very inefficient as it requires the enumeration of all possible worlds. Therefore, the actual implementation works directly on the compact representation and avoids enumeration of possible worlds. Since we are studying *qualitative* effects in this paper, it is strictly speaking irrelevant how this is accomplished; it suffices to know that query results

conform to this semantics. In this section, we nevertheless present the main principles behind an algorithm for efficient querying of probabilistic XML.

Observe that each possible world is obtained by making one particular combination of choices for all choice points. An XML node n is only a member of a particular world iff for each ancestor probability node, n is a descendant of the chosen possibility node. Consequently, the probability for the existence of n can be calculated by multiplying the probabilities of all ancestor possibility nodes of n. The possibility of *co-occurrence* of two nodes in some possible world can be determined by finding their lowest common ancestor probability node n' and verifying that both nodes are descendants of the same possibility child of n' [vK08].

XPath is a navigational language. An XPath query can be seen as a specification of which 'walks' are allowed, i.e., which axis steps can be taken, which kinds of nodes can be visited, etc. The result of the query consists of the end points of all allowed walks. The semantics of an XPath query on a *probabilistic* tree is the union of its result in all possible worlds. In other words, it consists of the end points of all allowed walks in all possible worlds.

Consider which walks are taken if we execute an XPath directly on the compact representation while properly stepping over the probability and possibility nodes. If all nodes it visits are member of some particular possible world, then it is an allowed walk in that world, hence its end point is member of the union. If it visits two nodes that cannot co-occur in any world, it is a walk in no possible world and its end point should not be taken into account for the union. In short, the result of an XPath query consists of the end points of all allowed walks for which all nodes it visits can co-occur.

Consider, for example, the query /actors/actor[role='Lydia' and role='Lydia Ratliff']/name executed on Figure 7. The answer is empty in all possible worlds (see Figure 6). Without the co-occurrence condition, executing this query directly on the compact representation produces some answers. The associated walks, however, visit both role nodes on the bottom right which cannot co-occur.

For an XML database such as MonetDB/XQuery, the abovedescribed operations are highly efficient that require almost no data access and can easily be processed in bulk due to loop-lifting [BGvK+06].

To obtain an answer in the per-element answer style, we subsequently need to determine for each result value

1. its possible multiplicities, and
2. the probabilities of these multiplicities.

Both can be obtained with a simple recursive algorithm in which the aforementioned principles for co-occurrence and probability calculation are applied.

3.5 Handling user feedback

As we explained in [dKvK07b], the goal of user feedback is to improve quality by updating the database according to the feedback. As an example, one can imagine an address book on a PDA that will return phone numbers for contacts. The address book may be the result of data integration with other address books (possibly from other people) and hence be uncertain. After dialing a most likely number, the user can indicate that he indeed talked to the person indicated in the address book (*positive feedback*), or that the person he talked to was not the one indicated in the address book or that the number is invalid (*negative feedback*). In general, the user indicates that one or more possible answers in the query result (in case of positive feedback) *do* or (in case of negative feedback) *do not* correspond with his knowledge of the real world. Based on possible world theory, a semantically correct way of handling such feedback is by eliminating all possible worlds that disagree with the statement on the query result. Note that this defition of user feedback is rather merciless. We assume that a user only issues feedback if he/she knows it to be absolutely and indisputably true. How to benefit from user feedback that itself can not be fully trusted is future research.

Definition 4 (feedback types) Let $v^m \in \overline{\mathsf{Ans}}_Q(PD)$ be a possible query answer for some query Q. Negative feedback is a statement "$m = 0$" for some value in the per-element answer style. The meaning of this statement is that the answer does not occur in the real world, i.e., $v \notin Q(\mathsf{RW}_{user})$ where RW_{user} represents a user's knowledge of the real world. Analogously, positive feedback is a statement "$m \geq 1$" meaning $v \in Q(\mathsf{RW}_{user})$.

Definition 5 (feedback effect) Let $S = \{(p,D) \in PD \mid v^m \in \overline{\mathsf{Ans}}_Q(PD) \wedge F\}$ where $F \equiv (m = 0)$ or $F \equiv (m \geq 1)$. S represents the set of possible worlds that pass the user feedback "$m = 0$" or "$m \geq 1$", respectively, for some v^m and query Q. Note that they still have their original probabilities. In [dKvK07b] it is explained that simple normalization is the correct way of recalculating the probabilities according to possible world theory. The probabilistic database after feedback PD' can now be defined as $PD' = \{(\frac{p}{G}, D) \mid (p,D) \in S\}$ where $G = \sum_{(p,D) \in S} p$.

As with querying, the semantics of user feedback is defined in terms of possible worlds and a naive implementation following this principle is very inefficient. Therefore, an actual implementation should work on the compact representation directly. A *generic* algorithm for efficient application of user feedback is still an open issue. We have implemented an efficient but limited user feedback algorithm which is restricted to feedback on XPath queries without predicates and to answers that have a multiplicity of 1. Both restrictions

have to do with being unable to determine from the feedback statement which nodes are wrong. Figure 9 gives an example of a tree for which query '//a' results in an answer with multiplicity 2. If the correct multiplicity of 'a' is 1, it is unknown whether a_1 or a_2 is the wrong one. The correct result should reflect 3 possible worlds, i.e., excluding the one with both a's. Note that user feedback may still discover the truth here: if it gets positive feedback on either b or c. If an answer to a predicate query is wrong, it is unknown if it is due to the node satisfying the predicate or the result node.

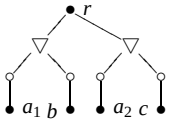


Fig. 9 Example of feedback on high multiplicity answer

Within these restrictions, negative feedback on an answer element v can be applied by finding all text-nodes with value v and their lowest ancestor possibility nodes. These possibilities produce value v so must be wrong, we cut them out and normalize the probabilities of the remaining possibility nodes. If a probability node ends up without children, then the next lowest ancestor possibility node must be wrong. We do this recursively up the tree. Positive feedback on an answer element v is applied by checking level by level whether a possibility node has a descendant node with value v, hence is capable of producing the answer v. If not, it must be wrong and is analogously cut out.

Note that our implementation may not be able to achieve maximum improvement for all feedback, but it conforms to the above semantics in that no possible world is incorrectly eliminated; it may only happen that some possible worlds that could have been eliminated, are not.

Promise for a more generic and efficient solution to handling user feedback can be found in [KO08] which presents an approach for conditioning probabilistic relational data in MayBMS based on additional evidence. Their algorithms work on a succinct representation of the remaining possible worlds in the form of a *ws-set*. A *ws-tree* is constructed for the purpose of efficiently adapting (conditioning) the probabilities of the tuples in the database. Although XML data is of a rather different nature, there are many correspondences which makes the problem similar. Probability nodes can be seen as MayBMS's random variables, possibility nodes their individual assignments, hence the '*ws-set* of a probabilistic XML document' is already present and organized in a tree. We plan to investigate how adapt this approach to probabilistic XML data. The main problem seems how to restructure an already existing *ws-tree* based on new evidence, because ordinarily there are many dependencies in probabilistic XML due to ancestor-descendent relationships, hence constructing a completely new *ws-tree* every time is bound to become a serious bottleneck.

$\mathsf{matches}(\mathsf{A},\mathsf{B}) = \{\langle \mathsf{a}=c_a, \mathsf{b}=c_b, \mathsf{e}=est\rangle | c_a \in \mathsf{A} \wedge c_b \in \mathsf{B}$
$\wedge\, est = \mathsf{Oracle}(c_a,c_b) \wedge est > 0\}$
$\mathsf{growCluster}(\mathsf{C},\mathscr{M}) = \text{IF } \mathsf{GC}=\mathsf{C} \text{ THEN } \mathsf{C} \text{ ELSE } \mathsf{growCluster}(\mathsf{GC},\mathscr{M})$
$\text{WHERE } \mathsf{GC} = \{m \in \mathscr{M} | \exists c \in \mathsf{C} : m{\cdot}\mathsf{a} = c{\cdot}\mathsf{a} \vee m{\cdot}\mathsf{b} = c{\cdot}\mathsf{b}\}$
$\mathsf{cluster}(\mathscr{M}) = \text{Take arbitrary } m \in \mathscr{M}$
$\mathsf{C} := \mathsf{growCluster}(\{m\});\ \mathsf{R} := \mathscr{M} \setminus \mathsf{C}$
$\text{RETURN IF } \mathsf{R} = \emptyset \text{ THEN } \{\mathsf{C}\} \text{ ELSE } \{\mathsf{C}\} \cup \mathsf{cluster}(\mathsf{R})$
$\mathsf{combinations}(\mathsf{C}) = \{S \cup \{a \in \mathsf{A} | \neg\exists m \in S : m{\cdot}\mathsf{a} = a\}$
$\cup \{b \in \mathsf{B} | \neg\exists m \in S : m{\cdot}\mathsf{b} = b\}$
$| S \subseteq \mathsf{C} \wedge \forall m_1, m_2 \in S :$
$(m_1{\cdot}\mathsf{a} = m_2{\cdot}\mathsf{a} \vee m_1{\cdot}\mathsf{b} = m_2{\cdot}\mathsf{b}) \Rightarrow m_1 = m_2\}$
$\text{WHERE } \mathsf{A} = \{m{\cdot}\mathsf{a} | m \in \mathsf{C}\}, \mathsf{B} = \{m{\cdot}\mathsf{b} | m \in \mathsf{C}\}$

```
integrate(a, b) =
  IF a and b are text nodes
  THEN IF a = b THEN RETURN a
       ELSE RETURN ⟨prob⟩⟨poss prob=0.5⟩a⟨/poss⟩
                   ⟨poss prob=0.5⟩b⟨/poss⟩⟨/prob⟩
  /* Matching phase */
  M := matches(a/child::*, b/child::*)
  cert := {m ∈ M | m·e = 1}
  possible := {m ∈ M | ¬∃m' ∈ cert : m'·a = m·a ∨ m'·b = m·b}
  /* Clustering phase */
  clusters := cluster(possible)
  /* Result construction phase */
  result := NEW ELEMENT "same name as a and b"
  FOREACH e ∈ a/child::* WHERE ¬∃m ∈ M : m·a = c
    DO INSERT e INTO result
  IF full integration (not only enrichment)
  THEN FOREACH e ∈ b/child::* WHERE ¬∃m ∈ M : m·b = c
         DO INSERT e INTO result
  FOREACH m ∈ cert DO INSERT integrate(m·a, m·b) INTO result
  FOREACH C ∈ clusters DO
    prob := NEW ELEMENT "prob"; INSERT prob INTO result
    FOREACH comb ∈ combinations(C) DO
      poss := NEW ELEMENT "poss"; INSERT poss INTO prob
      FOREACH m ∈ comb DO
        IF m is a pair
        THEN INSERT integrate(m·a, m·b) INTO poss
        ELSE INSERT m INTO poss
  RETURN result
```

Fig. 10 Integration algorithm

3.6 Integration Algorithm

The algorithm for our probabilistic data integration approach is given in Figure 10. The algorithm is basically a tree merge algorithm where subtrees belonging to entities which possibly refer to the same rwo are recursively merged for each possible combination of matches. Any data associated with these entities that only occurs in one of the data sources is simply added to the entity trees. In this way, we can supplement or enrich the entities with data from other sources even if the entities in different sources cannot be matched with certainty. A configuration parameter determines for which entities we are interested in a full integration (default) or in *enrichment* of data for only those entities present in a particular source. Thus configured, data enrichment is non-associative. In our example application, we configured only data enrichment for movie entities and full integration for everything else.

Note that this algorithm is an improvement of the one presented in [KKA05] in that it results in a more compact integrated result because of the additional clustering phase. The result construction phase in the algorithm is linear in the number of clusters and exponential in the size of the cluster.

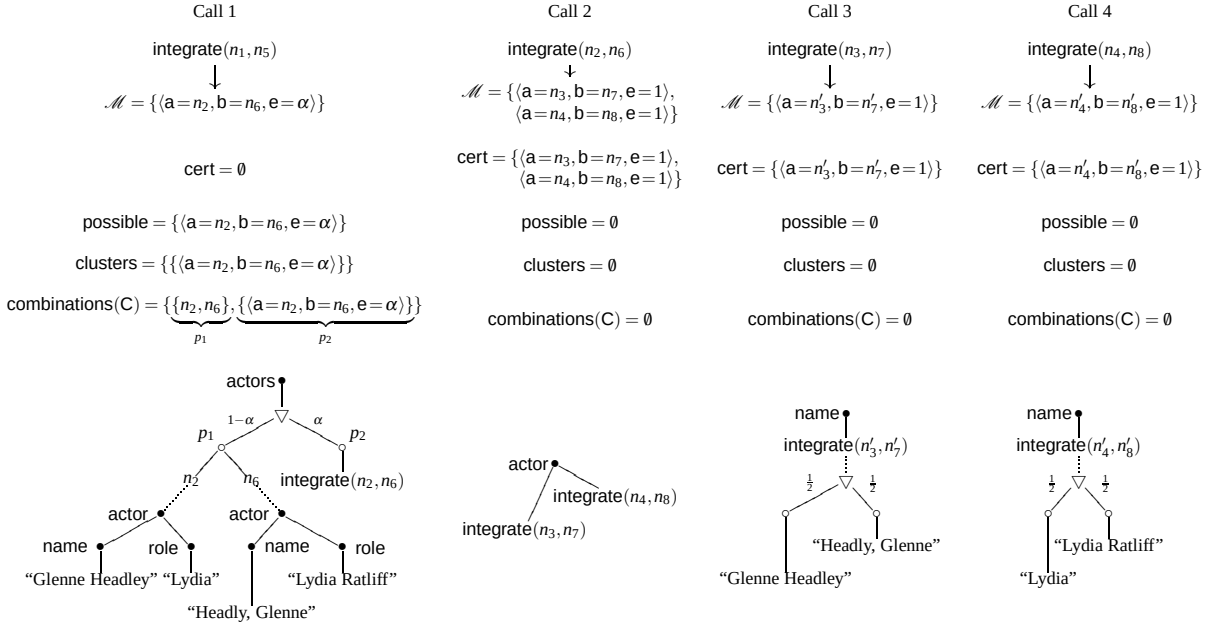


Fig. 11 Example execution of integration algorithm

The algorithm in [KKA05] in a sense views all matches as belonging to one cluster, hence was far from scalable. In this section, we explain the algorithm in general terms and illustrate it by explaining its behavior for the example elements given in Figure 5. Details of this example run are given in Figure 11. Note that in Figure 5, we didn't depict text nodes as separate nodes. Here we do and we denote the text node under node n_i with n'_i.

The inputs to the algorithm are two XML elements a and b. We assume that their schemas are aligned. Integration is performed recursively on a level-by-level basis. This can be seen in the calls to integrate within the integrate function itself. Its arguments are always $m{\cdot}$a and $m{\cdot}$b which are children of the input a and b respectively. The algorithm first checks if the input nodes are text nodes, because then it can immediately return a result. We left out the recursive calls for text nodes in Figure 11. In our example run, we start with 'Call 1' executing integrate(n_1,n_5). During execution, integrate is called recursively three more times (Calls 2–4).

Each recursion step is divided into three phases.

Matching phase. First we need to find possible matches between the children of the two input elements. Matching is performed by calling a function Oracle for each possible pair of children. It estimates the similarity of two given elements based on a set of knowledge rules. We discuss these knowledge rules in Section 3.7. The Oracle is sure to return an estimate of 0 when the given two elements have a different element name. In this way, it does not matter for our algorithm if the children it is matching are all of the same type (e.g., in Call 1) or of mixed types (e.g., in Call 2).

Some matches have similarity 1, which means that the Oracle is certain that they match. We distinguish those in cert from the rest, because if there is a certain match for a child, then we need not consider other matches of lower similarity for this element. The remaining matches in possible represent all entities for which we cannot make an absolute decision. Therefore, for each of these we need to consider both the possibility that they refer to the same rwo and the possibility that they do not.

If invoked on the `actors` elements of Figure 5 (Call 1), the Oracle estimates the similarity between the two `actor` children and returns α. Because $\alpha \neq 1$, its match ends up in possible and cert remains empty. In Call 2, the Oracle produces 4 estimations: two are 0 because name $\neq$ role and two are 1 assuming that there is a DTD restricting actors to have only one name and role. Both matches end up in cert and possible remains empty. Since no two text nodes can occur as siblings, the same happens in the Call 3 and 4.

Clustering phase. The set of uncertain matches usually contains small independent clusters of similar entities. For example, movies from IMDB similar to "The Hustler" are usually not similar to "Stage Beauty". It is beneficial to find these clusters to get a compact representation of the end result. Note that similarity is not transitive: it may occur within a cluster that data item A may be similar to B, B may be similar to C, but that A is not similar (enough) to C. Contrary to most entity resolution techniques, we properly handle this situation (see result construction below). To continue our example, only Call 1 has a non-empty possible. Since there is only one match, it produces one cluster.

The improvement of the algorithm of Figure 10 over the one presented in [KKA05] lies in this additional clustering phase. In [KKA05] we observed that the number of combinations increases dramatically with a rising number of possible matches. For example, if both a and b have 5 children and they all possibly match, then we end up with 1546 combinations, hence 1546 possibility nodes. As we argued above, many small clusters can usually be found. As a consequence, we do not produce one probability node with a huge number of possibilities, but several probability nodes, one for each cluster, with each just a few possibilities.

Result construction phase. Now we are ready to construct the result. We create one element which represents the integration result. All children of a that are not matched, are added to the result (this does not happen in our example). In case of full integration as opposed to enrichment, the same applies to children of b. For our TV guide example, we are interested in enrichment of the movies of the TV guide with data *only on these movies* from IMDB. In deeper recursion steps, we are interested in full integration of all elements. Subsequently, all certain matches are integrated and added as children. Then, for all certain matches the result of the integration is added (Call 2–4). In Call 3 and 4, we happen to integrate two text nodes for which we depicted the results immediately below in Figure 11.

For each cluster of uncertain matches we are faced with several possibilities. Therefore, we create a `prob` element (and add it to the result) for each cluster. A cluster contains a set of matches that can either be correct (i.e., the elements indeed refer to the same rwo in reality) or incorrect. The choices for each match are independent. The only restriction is that if an element a is matched to an element b, then a cannot at the same time be matched with another element $b' \neq b$, because in the original data source, b and b' were individual elements representing different entities. The combinations function determines all possible combinations of correct/incorrect choices for the matches of a cluster. Correct choices represent possibilities where we assume the elements refer to the same rwo, so we integrate them. Incorrect choices represent possibilities where we assume the elements do not refer to the same rwo, so they end up as individual elements in the combination and eventually as individual children of the `poss` element. Note that only combinations of matches are considered that were initially estimated to be possibly matching by the Oracle; in this way, we properly handle the non-transitiveness of similarity matching mentioned earlier.

To keep the presentation of the algorithm in Figure 10 clear, we have omitted the calculation of the probability of a combination. It basically is the product of the probabilities of the choices made for the matches that led to the particular combination.

For Call 1 we had one cluster with one match. There are only two possible combinations p_1 and p_2. Therefore, we construct a `prob` element with two `poss` elements.

3.7 The Oracle and its Knowledge Rules

The invocation of the Oracle is the only point in the data integration algorithm where a semantical decision is being taken. In this way, we have strictly separated the integration *mechanism* from the integration *intelligence*. The Oracle obtains its intelligence from knowledge rules. We distinguish between *generic* and *domain-specific* knowledge rules. The current set of generic rules in IMPrECISE is discussed in Section 3.8. Domain-specific rules are defined by the developer to enable the Oracle to produce good estimations by taking into account the specificities of the application domain. The role and effect of the knowledge rules in the Oracle can be understood best by imagining a few hypothetical 'extreme' Oracles:

- *Omniscient Oracle*. This Oracle has perfect knowledge, hence can always give a correct absolute estimate of 0 or 1 for each pair of elements. This is of course a hypothetical situation, but if we would be able to define all required knowledge rules for it, then cert would contain all positive matches and possible would always be empty. Therefore, we obtain only clusters of one match, so no probability and possibility nodes are constructed, and the algorithm produces an integration result without uncertainty.[5]
- *Ignorant doubtful Oracle*. This Oracle has no knowledge rules at all. For differently named elements it produces an estimate of 0; otherwise it produces 0.5 effectively stating that it is always fully in doubt. With this Oracle, all pairs of children would match, hence cert is empty and possible contains a cartesian product of all children. These matches all form one cluster which produces a huge number of possible combinations. Consequently, with this Oracle we get a maximally exploded integration result. Note that without any knowledge, the algorithm does work. The drawback is data explosion, because it considers all (non-sensical) possibilities and the quality of the integration result is low, because everything is equally likely.
- *Ignorant decisive Oracle*. This Oracle also has no knowledge rules, but stubbornly estimates all matches with 0. This is an interesting case, because this leaves cert and possible empty, hence the integration result contains a union of the children of both elements. This is

[5] Strictly speaking this is not true. In the presented algorithm, the Oracle does not make decisions about which value to take if data between sources conflicts. Text nodes always receive a 50/50 decision, so probability and possibility nodes are constructed for text nodes.

what is frequently done in practice to get an initial integrated data set, which is subsequently cleaned. Entity resolution happens in the data cleaning phase. Note that with data cleaning solutions, an absolute choice is always made for two data items to be 'duplicates' or not.

We start with the Ignorant doubtful Oracle. To obtain an integration that is good enough for a particular application, we add as many knowledge rules as needed to obtain an integration result that balances size of integration result with query answer quality. IMPrECISE contains a basic set of generic rules, so the developer needs to only define domain-specific rules that partially override the generic rules. We present the generic rules below. The domain-specific rules for our example application are given in Section 5.1.

3.8 Generic rules

Since we do not focus on similarity measures, but on how to cope with their inherent imperfections, we have implemented a simple edit distance measure that regularly gives too little evidence for an absolute decision. Unless overridden by other rules, two elements are compared based on *inverse relative edit distance* (red) of their string values:

$$\mathsf{red} \;=\; 1 - \frac{\mathsf{editdistance}(a,b)}{\mathsf{max}(\mathsf{length}(a),\mathsf{length}(b))}$$

Because of different naming conventions for names of things and people, e.g., "Kal Penn" and "Penn, Kal" in Figure 4, we count the switch around the comma in the edit distance as two edits (one delete and one insert). To be able to force absolute decisions, there are two thresholds:

1. A minimum threshold: if the red is below this threshold, The Oracle concludes with certainty that the two elements *do not* refer to the same rwo.
2. A maximum threshold: if the red is above this threshold, The Oracle concludes with certainty that the two elements *do* refer to the same rwo.

For any red between the two thresholds, the two possibilities are considered separately: the data items either refer to two different or to the same rwo. The red is taken as probability for the case that they do refer to the same rwo. The default min and max thresholds are 0.2 and 1.0 respectively (i.e., there is no max threshold). Certain paths can be configured with different thresholds, but this is domain-specific (Section 5.1 describes the configuration in our experiments).

Furthermore, elements with different element names cannot possibly refer to the same rwo, because the schemas are assumed to be aligned. Also, constraints on the schema imposed by a DTD are taken into account, e.g., if a certain element can only occur once and both data source contain the element, then they must refer to the same rwo, because otherwise we would end up with two elements in the result which is against the DTD.

4 Measuring Uncertainty and Quality

In this section, we define measures that quantify the intuitive notions of 'uncertainty' and 'quality' of integrated data. We define quality as the degree in which the data corresponds with "the truth" (the real world). The amount of uncertainty is an indication of how much system "doubts" its own data.

Measuring quality means a comparison with the truth, i.e., what a human with perfect knowledge would claim. Manual assessment of the quality of an entire database is too labor-intensive. Therefore, we measure the quality of query answers as an indication for the quality of the database. Querying uncertain data results in answers containing uncertainty. Therefore, an answer is not correct or incorrect in the traditional sense of a database query. We therefore define a more subtle notion of answer quality.

4.1 Measuring the amount of uncertainty

Number of possible worlds. An often used measure for the amount of uncertainty in a database is the number of possible worlds it represents. However, this measure exaggerates the perceived amount of uncertainty, because it grows exponentially with linearly growing independent possibilities. We therefore do not use it.

Uncertainty density and decisiveness. In [dKvK07a], we defined two other measures to quantify the uncertainty in the integration result in a query-independent way: uncertainty density (Dens) and decisiveness (Dec). The uncertainty density is a measure for the average number of alternatives per choice point. Note that this measure does not depend on the probabilities of the alternatives. On the other hand, even if there is much uncertainty, if one possible world has a very high probability, then any query posed to this uncertain database will have one, easy to distinguish, most probable answer. We say that this database has high decisiveness. For example, a very low or high α as opposed to $\alpha = 0.5$ in Figure 7 means answers to queries will on average have very low or high probabilities, hence it is easier to distinguish the more probable from the less probable answers. Both measures are defined as follows:

$$\mathrm{Dens} = 1 - \frac{1}{N_{cp}} \sum_{j=1}^{N_{cp}} \frac{1}{N_{poss,j}}$$

$$\mathrm{Dec} = \frac{1}{N_{cp}} \sum_{j=1}^{N_{cp}} \frac{P_j^{max}}{(2 - P_j^{max}) \log_2(max(2, N_{poss,j}))}$$

where N_{cp} is the number of choice points in the data (i.e., probability nodes in IMPrECISE), $N_{poss,cp}$ the number of possibilities or alternatives of choice point *cp*, and P_{cp}^{max} the probability of the most likely possibility of choice point *cp*. In case an XML node is a direct child of another XML node,

```
entropy(n) =
  IF kind(n) = prob
  THEN e := 0
       FOREACH poss IN child(n)
         p := prob(poss)
         e := e − p log2 p + p ( Σ_{c∈child(poss)} entropy(c) )
       RETURN e
  ELSE RETURN ( Σ_{c∈child(n)} entropy(c) )
```

Fig. 12 Algorithm for computing the entropy

we treat it as a choice point with one alternative, i.e., it counts as 1 in N_{cp}.

Entropy. "In information theory, entropy is a measure of the uncertainty associated with a random variable. The term by itself in this context usually refers to the Shannon entropy, which quantifies, in the sense of an expected value, the information contained in a message, usually in units such as bits. Equivalently, the Shannon entropy is a measure of the average information content one is missing when one does not know the value of the random variable. [...] Information entropy and information uncertainty can be used interchangeably." [WikiPedia].

For a random variable X with n possible outcomes x_i, the entropy is defined as $-\sum_{i=1}^{n} p(x_i)\log_2 p(x_i)$ where $p(x_i)$ is the probability of outcome x_i. In the possible worlds setting, an outcome x_i represents a possible world, hence

$$\mathsf{E} = -\sum_{T \in \mathsf{PWS}_{PT}} \mathrm{P}(T \mid PT)\log_2 \mathrm{P}(T \mid PT)$$

Although entropy is defined in terms of possible worlds, it is not necessary to enumerate all possible worlds to compute it. Figure 12 contains an algorithm for computing entropy based on a recursive descent of the probabilistic XML tree. Appendix A contains a proof that what the algorithm calculates is indeed the entropy.

4.2 Answer Quality

In the possible world approach, an uncertain answer represents a set of possible answers each with an associated probability. In some systems, it is possible to work with alternatives without probabilities, but these can be considered as equally likely alternatives, hence with uniformly distributed probabilities.

The set of possible answers ranked according to probability has much in common with the result of an information retrieval query. We therefore base our answer quality measure on precision and recall [BYRN99]. We adapt these notions, however, by taking into account the probability with which a system claims a query answer to be true. The intuition behind it is that the quality of a correct answer is higher if the system dares to claim that it is correct with a higher probability. Analogously, incorrect answers with a high probability are worse than incorrect answers with a low probability. We believe that taking into account the probabilities provides a better retrieval quality measure than traditional precision/recall measures with which many other approaches in this field are being evaluated (e.g., [SDH08]).

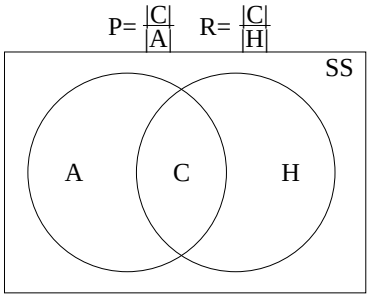


Fig. 13 Precision and recall.

Precision and recall are traditionally computed by looking at the presence of correct and incorrect answers. Since XPath and XQuery answers are sequences and we ignore order, we define H to be the *multiset* of correct answers to a query (as determined by a human), A the multiset of answers produced by the system, and C the intersection of the two, i.e., the multiset of correct answers produced by the system (see Figure 13).

Since we are in a probabilistic setting, it is logical to take the *expected value* of the precision and recall. This naturally takes into account the probabilities associated with answers. Let E denote the expected value. We define A as $[\{Q(PD)\}]$, C as $A \cap H$, and for any multiset with probabilities S, the expected cardinality as $E(|S|) = \sum_{(v^m,p)\in S} p \times m$. Then,

$$E(\text{Precision}) = \sum_{(p,D)\in PD} p \times |\text{Precision}_{Q(D)}| = \frac{E(|C|)}{E(|A|)}$$

$$E(\text{Recall}) = \sum_{(p,D)\in PD} p \times |\text{Recall}_{Q(D)}| = \frac{E(|C|)}{|H|}$$

In essence, expected precision assesses the ratio of *probability mass* of correct answers w.r.t. the probability mass of the complete query result. Recall assesses the total probability mass of correct answers in the query result. For example, suppose the answer to the query "Give me all movies aired on CMAX on June 5" is "The Namesake" and "Namesake, The". We consider textual variations of the same semantical concept as the same answer. If the system returns this single answer (which is correct), but with a confidence of 90%, then precision and recall are both 90%. If, however, it also gives some other (incorrect) movie with a confidence of 20%, precision drops to 82% and recall stays 90%.

Any measure derived from precision and recall, such as the F-measure, can be adapted analogously. An alternative answer quality measure is *query reliability* [dR95, GGH98]. This measure is based on the Hamming distance between

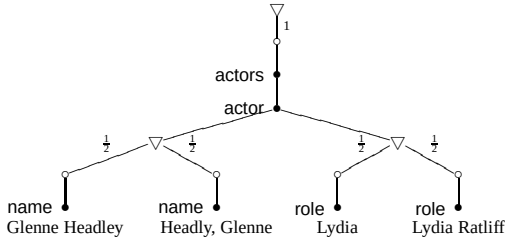


Fig. 14 Example with higher uncertainty than Figure 7, but also higher quality

the answer and the correct answer, i.e., the number of tuples (or elements) where the answer differs from the correct answer. Query reliability also does not take into account the probabilities of the elements in the query answer. One can imagine an analogous extension of query reliability where the sum of differences in probability mass for all elements in the answer is determined. In the sequel, we use our measure of expected precision and recall.

4.3 Interpretation of uncertainty measures

Note that uncertainty and quality seem to be related in the sense that less uncertainty seems to indicate higher quality. Our definition of the terms, however, make them truly orthogonal notions. Compare for example Figure 7 with Figure 14. The latter has higher quality, because it is closer to the truth. Observe, however, that at the same time it has *relatively* more uncertainty, because an entire subtree with many certain nodes is not present. The measures for uncertainty of Figure 7 vs. Figure 14 support this fact: ($\alpha = 0.3$) Density 0.15 vs0.25, Decisiveness 0.82 vs. 0.67, and Entropy 1.48 vs. 2. Intuitively speaking, in Figure 7 the system rather firmly believes many facts to be highly likely but this believe is simply wrong, hence although the system contains less doubt/uncertainty than in Figure 14, the data is of lower quality. It is important to understand the difference in meaning between uncertainty and quality while interpreting the experimental results.

5 Rule definition and threshold tuning experiments

5.1 Experimental set-up

The aim of the experiments is threefold:

1. to check our assumption that a proverbial 90% of the entities are easy to match with simple matching rules,
2. to investigate the sensitivity of rule definition and threshold tuning on integration quality, and
3. to use this insight as evidence that our probabilistic integration approach indeed significantly reduces development effort.

By defining knowledge rules and thresholds, a developer aims to get rid of as many incorrect possibilities as possible to reduce the uncertainty and consequently the size of the integration result, but at the same time to not run too much risk in ruling out correct possibilities. Therefore, the following factors play a role in the experiments.

- Knowledge rules.
- Thresholds.
- Size of the integration result.
- The amount of uncertainty in the integration result.
- Quality of answers to certain queries.

The domain of the experiment concerns movies. In the integration, only the entities ‘movie’, ‘actor’, and ‘genre’ are present in *both* data sources, hence play a role in the entity resolution for this application. Therefore, the domain knowledge added to the system focuses on these entities. Figure 4 shows which other elements accompany these entities. Note that a correct match on ‘movie’ determines whether or not the entity is enriched with (correct) data from IMDB.

The knowledge rules. The domain-specific knowledge rules we define and play with are the following.

1. **DTD rule**: The DTD prescribes that certain elements occur only once among others title, year, and within the actor-element: name and role.
2. **MovieTitleYear rule** (MTY-rule): The probability of two movie elements referring to the same rwo is based on the year and the similarity of their titles. To obtain candidate matches for a particular movie title, we search for those movie titles that have the least edit distance. If the best edit distance is not zero, we expand the candidates with all titles within a margin of *n* additional edit distance. We further require that the year-attributes of two movie-elements need to be the same and that the red of the titles is above a certain threshold. The latter allows the rule to properly detect cases when there exists no matching movie. We use this rule as a representative candidate of a simple rule that a developer would typically write for matching movies.
3. **MovieCommonRoles rule** (MCR-rule): Opposed to the above two rules that match movies based on title and year information, this rule matches movies by looking only at the actors’ roles. It decides that two movie-elements match if the amount of actor roles they have in common is above a certain percentage. Because it is a rather computationally intensive rule, we use this rule only as a check that we did not miss any hard to find matches with the MTY-rule.
4. **UniqueRole rule** (UR-rule): If the role-child of two actor-elements is exactly the same and the role is unique for

the movie in both data sources, then the Oracle concludes with certainty that it is the same actor. Otherwise, the probability is computed by multiplying the red's of the name and role children which is subject to the actor threshold. As a consequence, the decision of The Oracle is solely based on the actor name, when the role is the same but not unique or the role is missing in one of the sources.

Note that the only development effort needed for the TV guide application to safely and sufficiently reduce the number of incorrect matches for entity resolution of movies and actors, hence to be able to meaningfully use the integrated data set, is the definition of the DTD, the definition of a subset of the above rules, and to tune the associated thresholds.

Parameters and thresholds. Generic and domain-specific rules usually contain parameters and thresholds as a means to force absolute decisions. The parameters that play the most prominent roles in the experiments are:

- (MTY-rule) Movie title *margin*. We vary margin *n* in the experiments. A higher *n* means running less risk of not finding the one correct match, but also means considering more possible matches, hence more uncertainty.
- (MTY-rule) Movie title *threshold*. We vary the minimum threshold on movie title similarity in the experiments. A lower threshold means running less risk of not finding the one correct match, but also means considering more possible matches, hence more uncertainty. The threshold is meant to properly dismiss too dissimilar matches when there exists no matching movie at all.
- (UR-rule): Actor *threshold*. We vary the minimum threshold on actors in the experiments. A lower threshold means running less risk of not finding the one correct match, but also means considering more possible matches, hence more uncertainty.

Data sets. We use for our experiments the application context of a TV guide application. As representative data, we extracted data from the `www.tvguide.com` website. The data set contains data from their top-100 'Most popular movies' page (which surprisingly contained 98 movies when we extracted it from the website).

We integrate with data from the IMDB movie database (version 3.24) publicly available from various FTP-sites with the aim to *enrich* the data of the TV guide with more details. We converted the data to XML with a schema aligned with the TV guide data. We excluded documentaries and adult movies to leave a realistic and substantial data set of 243,856 movies in a 253MB XML document. A perfect integration results in an integrated data set of 42311 nodes.

We like to highlight the following interesting properties in our data set.

- The two data sources follow a different convention for representing titles starting with "The". For example, "The Last Samurai" in the TV guide is called "Last Samurai, The" in IMDB. Our red measure estimates this as a similarity of 0.81. Since these matches are not perfect, the system starts to confuse these movies with other movies at lower thresholds or high margins. Most similar titles are 'Last American, The' and 'One Last Dance' at 0.5 and 0.43, respectively.
- There is a difference in convention with actor names as well. For example, "Paul Newman" is represented as "Newman, Paul (I)" in IMDB with similar consequences.
- The TV guide data is much newer than our IMDB data, so many new movies do not have a match in IMDB. Similarity-based matching unaware of this fact, will find many 'best' matches with similar scores. In particular movies with a short title, such as 'Bolt' and 'W.', are a challenge for the system.
- The top-100 data of the TV guide contains the movie 'Man of the House' (2005). There are, however, 2 movies called 'Man of the House' in 2005 in the IMDB data set. The MTY-rule is unable to distinguish these two movies at any margin and threshold.

Factors. We investigate threshold tuning sensitivity by varying the margin and minimum red threshold for titles in the MTY-rule and the minimum red threshold for actors that do not hit the UR-rule (i.e., no matching role or the role is not unique). Varying the MTY thresholds has no effect on movies with an exact match. If only an imperfect match for a movie exists, at low margin and high threshold no match is found, hence the system considers them as different movies and the TV guide data is not enriched with IMDB data. Increasing the margin or decreasing the threshold this does happen. Going further still means that probability mass going to the correct matches decreases. For movies and actors without a match, increasing the margin or decreasing the threshold only increases the confusion.

Finding no match for an actor means that the system regards them as different actors, hence the actor appears twice in answers to queries. Note that when an incorrect match for a movie is considered as a realistic possibility, the actor lists of both movies are integrated as well possibly producing some correct matches, but more likely producing a union of the actor lists when it determines that all actors are different people. At a high threshold for actors, correct matches are often not found. Figure 4 shows many cases where for the same real-world actors, names and roles differ substantially in their data of both sources. Decreasing the threshold will increase the likelihood that they are matched, but also the likelihood for incorrect matches.

To investigate the effect of rule definition, we switch off some of the DTD-rules (title, year, genres, directors, plots,

Group	#Queries	Query pattern
A	18	For each leaf tag name, an absolute path to that leaf
B	5	`//movie[.//genre='X']/title`
	5	`//actor[.//role='X']/name`
	5	`//movie[.//role='X']/title`
	5	`//movie[.//keyword='X']/title`
	5	`//movie[.//location='X']/title`
Total	43	
C	12	All actor queries, i.e., the two absolute paths for role and name from A and the 10 queries from B with a predicate on role

Fig. 15 Groups of experimental queries (for X we chose 5 different values that focus on different possible mistakes).

Entity	Total	Exact Match	None	Ambiguous	Wrong	Percentage easy cases
All (situation (b): many non-matches)						
Movie	98	56	27	14	1	84.7%
Actor	3887	1484	2101	302	?	92.2%
2006 and earlier (situation (a): few non-matches)						
Movie	57	48	2	6	1	87.7%
Actor	3133	1447	1400	286	?	90.9%

Fig. 16 Results of checking the proverbial 90% assumption.

and locations) and the UR-rule respectively and compare the quality of the answers to those where the rules were switched on. We also investigate the change in effect of threshold tuning.

Queries. We assess the quality of our integration result by taking the average expected precision and recall for different subsets of 43 queries (see Figure 15). The queries are chosen based on the following aspects:

- Group A makes sure we query all the data in the data set.
- Group B contains queries with predicates containing queries that are independent of entity matching problems in actors, or that are sensitive to the interplay between actors and movies, or that specifically target elements from either the TV guide or IMDB.
- Group C contains all queries that involve both movie and actor information for investigating the interplay between the movie and actor entities.

5.2 Results

In this section, we discuss the various effects of knowledge rule definition and threshold tuning on the integration quality. We first check our assumption that the proverbial 90% of the movies and actors are indeed easy to match. We then look at threshold tuning by investigating the effect of varying the movie title margin, movie title threshold and actor threshold. We finally investigate the influence of rule definition by switching off the DTD and UR rules.

The proverbial 90% assumption. In this experiment, we integrate the top-100 movies with the IMDB data set using the DTD-, MTY-, and UR-rules as representatives of simple rules a developer would typically define for such an application. A typical quick method for choosing rough values for thresholds, is to play with them on a sample. Doing that quickly learns that a rough setting of 2 for the title margin finds most matches if they exist, that a rough setting of 0.5 for the title threshold safely discards most false positives when movies have no match in IMDB, and that rough settings of 0.2 and 0.8 for minimum and maximum threshold on actor names safely finds all matches without producing unnecessary many false positives.

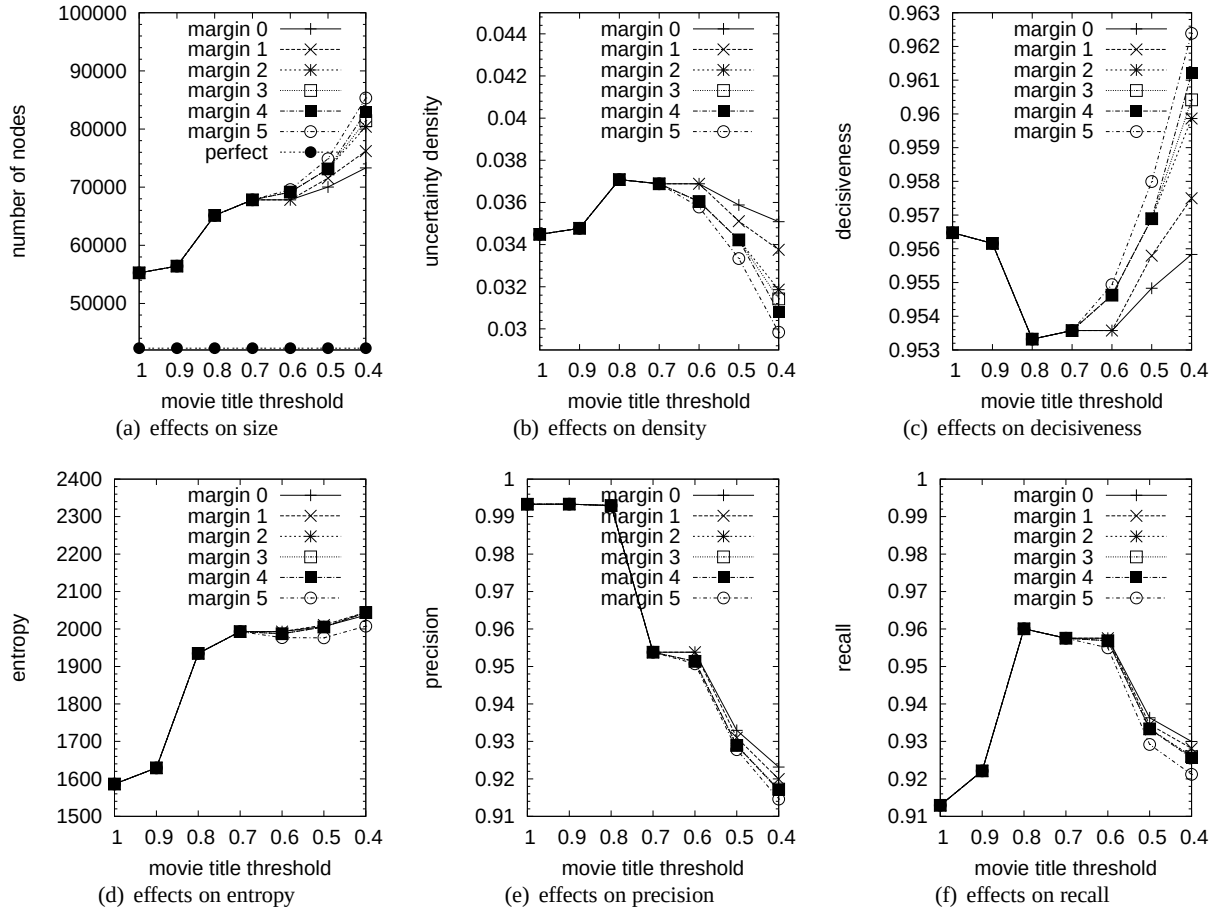
We measure the percentage of easy cases, i.e., for how many movies are an unambiguous and correct match is found or for which no unambiguous and correct match is found. To determine whether or not the matches are correct, we compared the resulting matches for movie entities with the result of matching under the MCR-rule. The rule uses other information (common role names) to match movies. We classify all exact matches and non-matches for actors as 'easy' and ambiguous matches as 'hard'.

In general, we typically encounter two distinct situations: (a) an entity almost always has a match, or (b) there are many non-matches. With our data, we can mimick both. The TV guide data is much newer than our IMDB data, so many new movies do not have a match in IMDB. Therefore, by doing the experiment with all movies, we have situation (b). By doing the experiment with only the movies of 2006 and earlier, we have situation (a).

Figure 16 shows the results of the measurements. It turns out that the above rules and thresholds correctly and unambiguously find matches for 56 movies, and determine correctly that 27 have no match in IMDB. For 14 movies, the system produced besides the correct match other matches, hence these constitute the hard ambiguous cases. One non-match turned out to be wrong: "Crouching Tiger, Hidden Dragon" (2000) has title "Wo hu cang long" in our IMDB data set. Of course no string matching algorithm can possibly find this match, but the MCR-rule can, because it does not depend on title information. In total, 84.7% of the movie entities can be said to be easy to resolve. For actor entities, 92.2% of the matches were unambiguous. The situation changes only slightly when we consider only movies of 2006 and ealier. Only 2 such movies have no match in our IMDB data set. The percentage of easy cases are 87.7% for movies and 90.9% for actors.

We can conclude that the assumption that 90% of the entities can easily be resolved roughly holds. It is not exactly 90% in our application, but it ranges from 85% to 92% depending on the circumstances for both entities.

We now turn our attention to analysing the sensitivity of our approach to threshold settings.



(a) effects on size (b) effects on density (c) effects on decisiveness

(d) effects on entropy (e) effects on precision (f) effects on recall

Fig. 17 Effects of MTY margin and threshold (actor threshold 0.4)

Effect of movie title margin and threshold. Figure 17 shows the results of varying the movie title margin and threshold while keeping the minimum actor threshold at 0.4. The expectation is that the size of the integration result increases with a larger margin and lower threshold as more matches are considered possible. Figure 17(a) shows that the size indeed increases from 55261 to between 73000 and 85000 depending on the margin. A perfect (manually constructed) integrated document holds 43211 nodes, so the additional uncertainty produces a document that is 1.28 to 1.97 times larger, perfectly manageble for a scalable XML database such as MonetDB/XQuery [BGvK+06].

At margin 0 and threshold 1, only the exact matches are found. Decreasing the threshold a bit (left side of the graphs) results in more matches to be considered possible. It increases the size and uncertainty density and lowers the decisiveness, but we gain especially in recall as more movies get enriched with correct data from IMDB. Precision remains high since under these circumstances most answers that are given are correct, because they either come from the TV guide (hence are mostly correct) or from IMDB (mostly belonging to correct matches). Entropy grows quickly as the number of possible worlds increases rapidly.[6]

It may seem in general counter-intuitive that data with more uncertainty can be of higher quality, but it is an important fact we like to draw your attention to: by considering more possible matches we find and incorporate more correct ones which increases recall, but this comes at the expense of considering more matches, hence dealing with more uncertainty. Our point is that this trade-off is worthwhile. We are interested in a 'good-enough' *initial* integration which is a suitable starting point for improvement by user feedback. The latter is effective in eliminating incorrect possibilities but cannot invent new data.

Further to the right in the graphs, most of the additional possible matches are wrong and subsequent integration of the actor lists is easy (no actors match for two different movies). As a consequence, at a lower threshold, there is more data with low ambiguity, so uncertainty density starts to drop again and decisiveness increases. The behavior of the entropy shows signs of two counteracting effects: a grow-

[6] As a matter of fact, we have stopped counting possible worlds, because the numbers go beyond the range of double $\gg 10^{308}$.

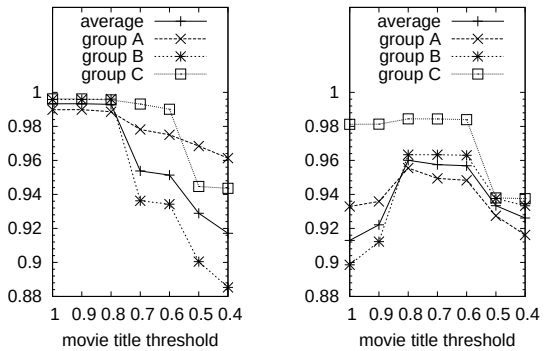


Fig. 18 Effects of query groups (actor threshold 0.4; margin 2)

ing number of possible worlds means a higher entropy and more extreme probabilities mean a lower entropy (see Figure 17(d)). Diminishing quality at higher margins and lower thresholds can be explained by effects on probabilities. Even though the extra possibilities that are considered now have low probabilities, they do 'consume' some probability mass, hence correct answers receive slightly lower probabilities.

It depends on the application whether precision is more important or recall. Since we are interested in an *initial* data integration, we are after good recall, perhaps at the expense of some precision. The expectation is that user feedback can quickly and effectively improve precision later during use of the application (see Section 6). User feedback can never 'repair' any missed match in the initial integration. A movie title threshold of 0.8 already provides the recall we want irrespective of the margin. Note that a developer does not have these curves at his/her disposal, so he/she does not know this. Most importantly though, if he sets the margin and threshold at a safe level, e.g., margin 2 or 3 and threshold 0.7 or 0.6, we see that recall does not change much, but he pays for this safety with some precision.As recall is considered more important, answer quality can be said to be rather insensitive to a moderately safe setting of the margin and threshold.

Density and decisiveness are based on the choice points directly. They don't incorporate that a node is a descendant of a choice point higher up in the tree. In a sense they only measure *local* uncertainty. Low density, high decisiveness, but at the same time low quality hence means that the data contains much locally certain but wrong data, for example actors and locations of incorrectly matched movies. Interestingly, the 'optimal' point with highest precision and recall co-incides with the highest uncertainty density, lowest decisiveness, and a flattening entropy. It is an open issue whether or not there is a correlation between these measures, but an important one to investigate as uncertainty metrics can be measured automatically and could aid in threshold tuning.

Figure 18 presents a closer look on what kind of queries are affected most. Querying actor information (group C) does well throughout, because the TV guide already has quite extensive information on actors so the enrichment with IMDB does not add much here. Moreover, queries with predicates (group B) are particularly sensitive to wrong thresholds. Behavior on some individual queries are worth noting. Query 41 `//movie[.//location[contains(.,'Las Vegas')]]/title` goes from 1 to 0 precision at threshold 0.7. The correct answer is empty (there is no movie with location 'Las Vegas'), but at 0.7 the system erroneously considers a match with a movie that contains this location in IMDB. The result of query 29 `//movie[.//actor/role='Rhino']/title` is 'Bolt' (2008). 'Bolt' has no match in IMDB and is erroneously matched with movies like 'Loot', 'Loft' and 'Goat'. For 'Loot' the actor 'Mark Polish' has no role description in IMDB and is confused with 'Mark Walton' at lower thresholds who happens to play 'Rhino' in 'Bolt'. As a consequence, quality drops considerably as 'Loot' enters the scene as a possible answer.

Effect of actor threshold. A difference between the actor and movie entities, is that an actor match *depends* on a movie match, because only the actors of possibly matching movies are matched. Moreover, with actors we do a full integration of the actor lists, i.e., actors that do not match are considered additional different actors. The UR-rule is able to make an absolute decision in many cases independent of the threshold. Only when the roles are not exactly the same or not unique within the movie, does the system rely on the threshold to determine whether or not we have a (possible) match.

Figures 19(a) to 19(d) show that varying the actor threshold displays the same behavior at the different movie margins and thresholds. Hence, the influence of the actor threshold can be said to be independent of the movie parameters even though actor matches depend on movie matches.

A second observation is that we see a steady increase in size, uncertainty density, entropy, precision and a steady decline in decisiveness. A reversal as observed in Figure 17 does not occur. Because we do a full integration, at high actor thresholds *all* actor names and roles are present in the query answers; they only may erroneously occur twice when the system does not consider then to be possibly the same. This is also the reason why recall is invariant. Contrary to the movie entity, at low thresholds no additional 'wrong' actors are matched, because actors are restricted to the ones belonging to the movie at hand. The only effect of lowering the actor threshold is that it matches more dissimilar entries, for example, the match 'Peter Gunn / Captain Laurent' and 'Gunn, Peter (I) / Capt. Laurent' is only found at threshold 0.4. These are often correct, so also precision keeps rising.

Very very low actor thresholds may create clusters of actors that the system cannot distinguish from each other. This only starts to happen in a very limited way at threshold

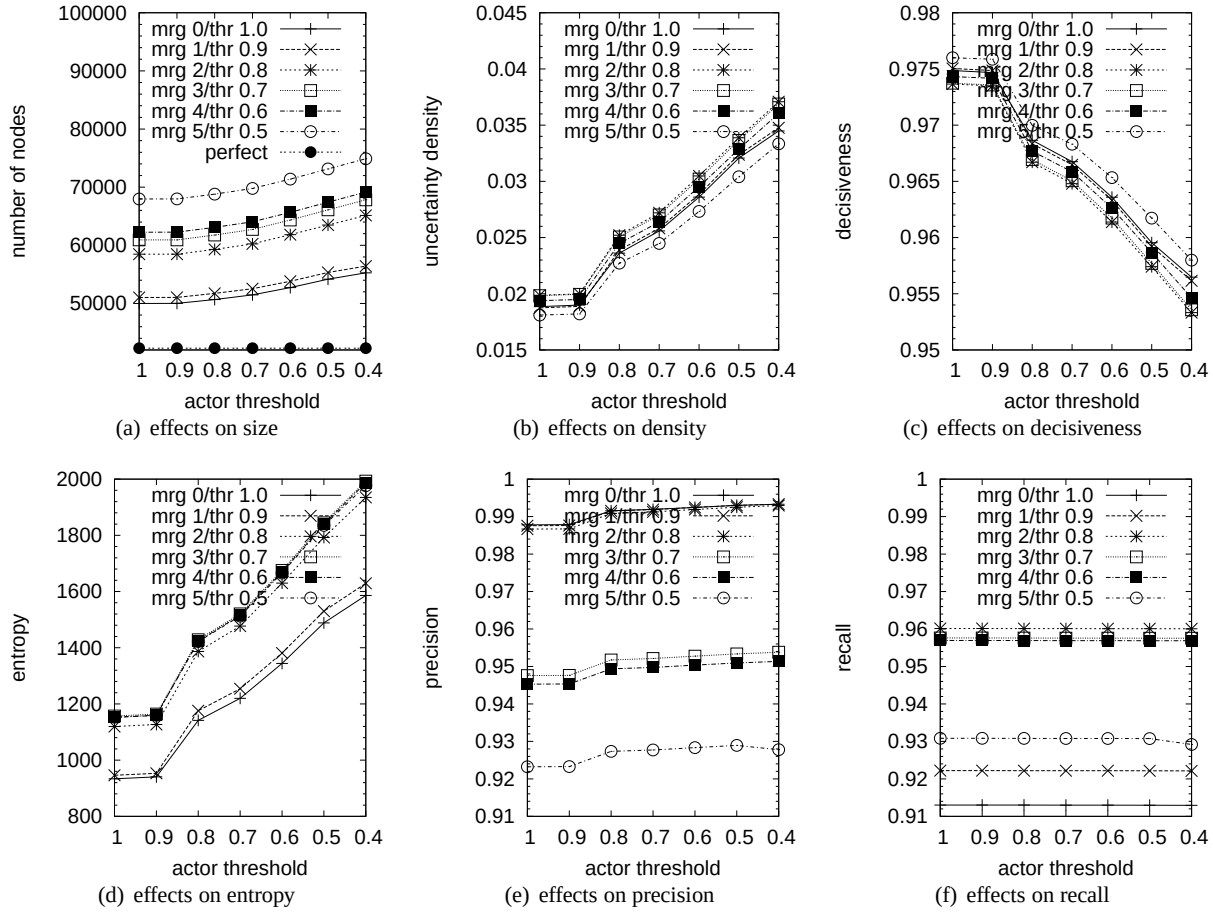


Fig. 19 Effects of actor threshold

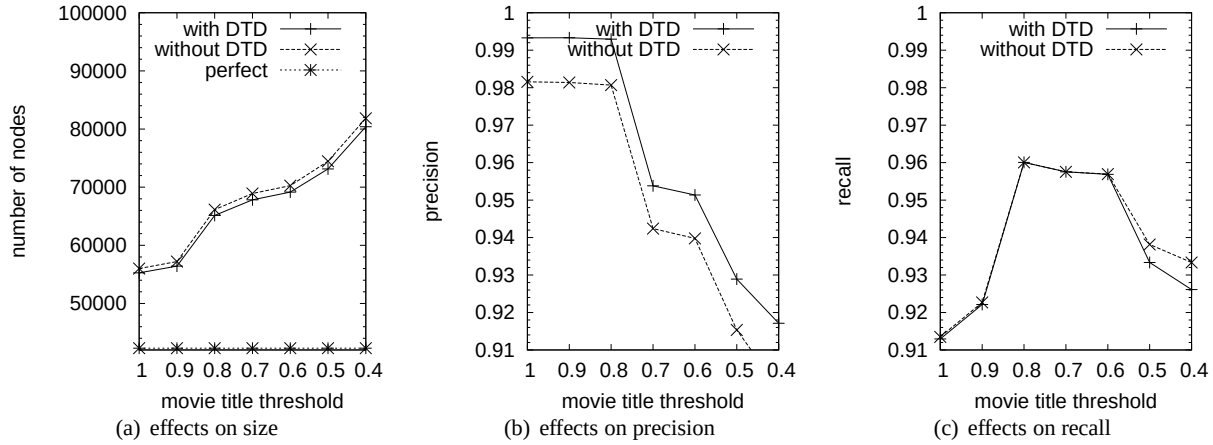
0.4. At 'mrg 5/thr 0.5' and actor threshold 0.4 where we see a small decline in precision and recall.

In general, we can conclude that for an entity which is dependent on another, quality is mostly determined by the other entity. We can safely choose a rather low threshold in order to not run the risk of missing a correct match. Note that this is only true upto the level that the actors of one movie can no longer be distinguished from each other. This happens when we switch off the UR-rule (see below).

Effect of DTD rules. Figure 20 shows the effect of switching off some of the DTD rules. A consequence is that the system no longer holds the domain information that title, year, genres, directors, plots, and locations can only occur once. The effects on the size, density, decisiveness and entropy is minor (only size shown). The effect on quality is mostly a constant drop in precision.

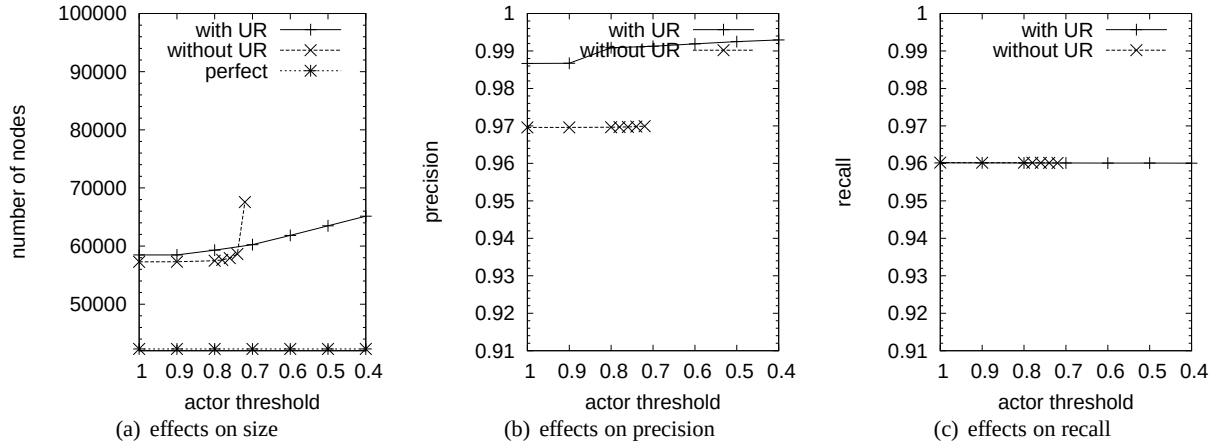
Effect of UR-rule. Figure 21 shows the effect of switching off the UR-rule. Because of the difference in naming conventions, the system only starts finding matches for actors at a threshold lower than 0.8. Because of the high number of actors in the data set, uncertainty quickly rises to unmanageable amounts: rightmost point is at 0.72; below that integration could not be completed. Consequently, it is important to have a rule like UR to guard against such an explosion of uncertainty when we are integrating a high volume entity. In general, we see a substantial drop in precision and no effect on recall similar to switching off some DTD rules.

Evidence for development effort reduction. The first observation that we like to emphasize, is that for each entity, in our case movie, actor and genre, we only had to specify *one* domain-specific rule to provide enough knowledge for roughly resolving the entities. For the rest the system relies on generic string matching and appropriate thresholds. The MTY-rule is far from perfect, it only relies on title and year for identification of movies, yet it is already able to achieve average precision and recall of around 0.99 and 0.96, respectively, with a safe title threshold setting between 0.8 and 0.6. (see Figure 17(e) and 17(f)). This is 'good enough' for an initial integration knowing that answers are accompanied with probabilities so the most probably ones can be easily picked out, and that feedback during use will gradually im-



(a) effects on size (b) effects on precision (c) effects on recall

Fig. 20 Effects of switching off some DTD rules (margin 2, actor threshold 0.4)



(a) effects on size (b) effects on precision (c) effects on recall

Fig. 21 Effects of switching off the UR rule (margin 2, movie title threshold 0.8)

prove the integration quality. The UR-rule makes an absolute decision for a large subset of actors, which effectively reduces uncertainty significantly.

Many improvements to the rule definitions are imaginable, but we refrained from doing so here to show that even with such preliminary rules, already a meaningfully usable integrated data set is obtained. The only real reason for spending time on improving rules is to prevent explosion of uncertainty like in Figure 21(a). Such obstructive growth only starts to occur under rather atypical conditions with stupid rules and extremely safe thresholds. The integration algorithm exhibits exponential result size growth with growing cluster size, but linear growth with growing numbers of clusters. Typically, only 10%-15% of the cases are indeed ambiguous and these typically form small clusters, hence growth in data size is not obstructive. For matching and clustering, many scalable algorithms are available.

The second important observation is that precision and recall are rather insensitive to moderate variations in threshold settings especially on side of a ‘safe setting’. With a safe setting we mean a setting that is not likely to miss any correct matches at the expense of considering quite some incorrect matches as realistic possibilities (low threshold for actor name or movie title). From this we conclude that a developer need not spend much time on threshold tuning. It suffices to just inspect matching results for a sample and then to choose a somewhat safer value.

In this way, our experiments have provided valuable evidence that our approach allows a developer who needs to integrate two data sources, to only focus on resolving the proverbial 90% of easy cases and save 90% of the work he would otherwise have to spend on resolving the 10% of hard cases. The only thing we need to subtract from the 90% development effort reduction is effort on rule definition and threshold tuning. For this, we just argued that few rules and rough settings suffice.

Some limitations in our experimental set-up need to be mentioned that call for further experiments and carefulness with drawing conclusions. First, we focused on one application and two fixed data sources. We believe that our re-

sults carry over to other contexts, but further experiments are needed to confirm that. Furthermore, our quality assessments are based on our newly introduced quality measure, expected precision and recall. We find it intuitively more attractive than other quality measures such as traditional precision and recall in that it naturally incorporates the probabilities that come with query answers, but comparative experiments are needed to show its strength.

Obviously, the remaining 10% of unresolved entities remain in the integrated data as uncertainty. As explained, we can defer their resolution to the end users of the integrated data using user feedback. Integration quality will gradually improve in this way. Most importantly, the application can be already be meaningfully used without resolving the 10% of hard cases, hence 'time-to-market' is reduced significantly. So, let us now focus on experimental evidence of the effectiveness of user feedback on integration quality.

6 User feedback experiments

6.1 Experimental set-up

The aim is to evaluate the effectiveness of user feedback on probabilistic data obtained from the integration of data sources. We take the integrated data obtained from enriching TV guide data at under the worst threshold conditions: margin 5, movie threshold 0.4 and actor threshold 0.4. On this data, we run a series of 100 consecutive feedbacks made by a simulated user who randomly picks a query and an element from the query answer and gives positive or negative feedback according to whether or not the answer is correct. Subsequently, we re-assess the average expected precision and recall for all 43 queries. To be able to conclude with general trends, we also run multiple shorter series with positive and negative feedback only. Because of algorithm limitations, we limit ourselves to giving feedback on the queries of group A only (the queries without predicate) and on answers that cannot have a multiplicity higher that 1.

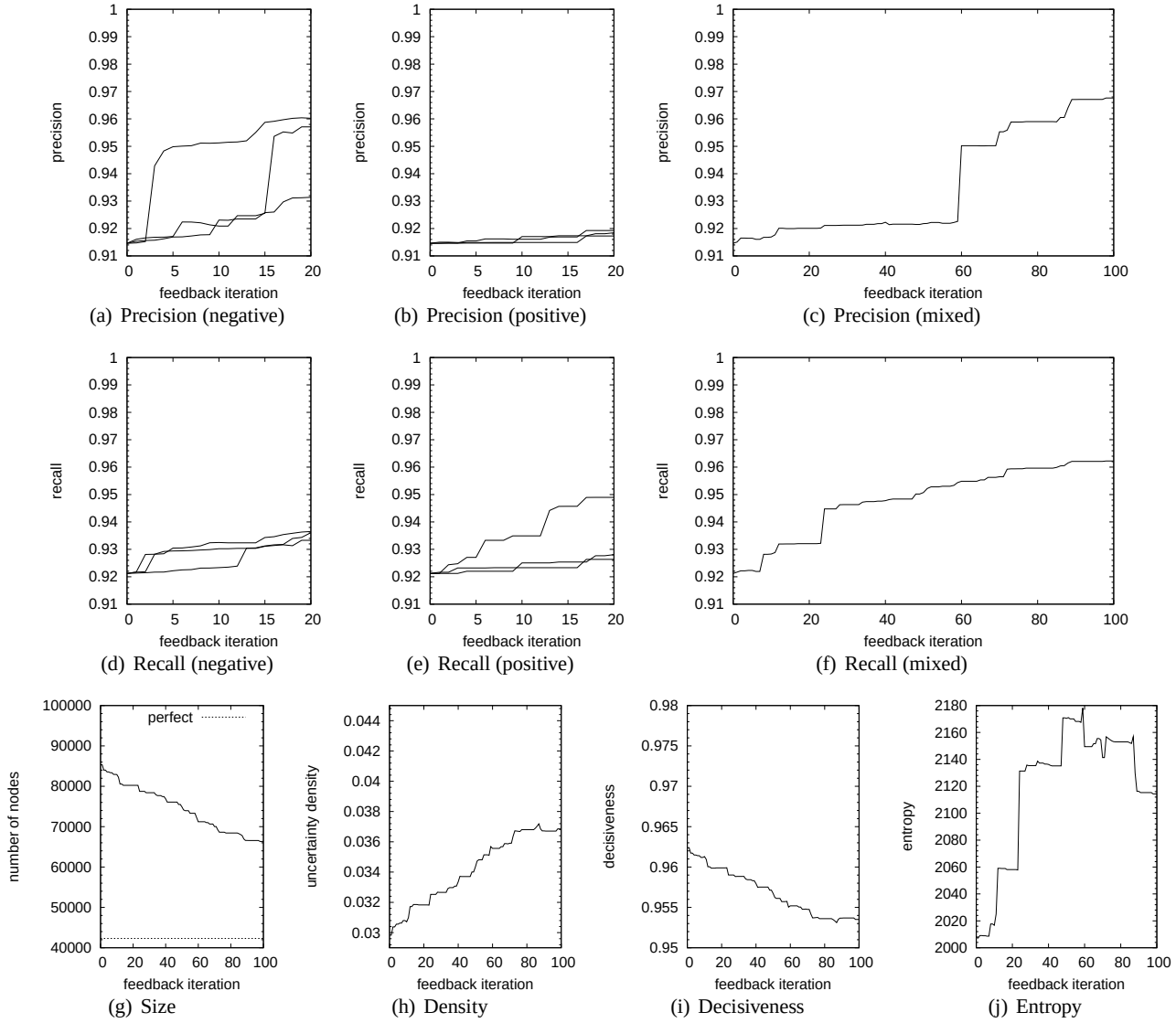
6.2 Results

Figure 22 presents the results. The top row shows precision, the middle row recall, and the bottom row the uncertainty measures. For precision and recall we show from left to right the series of 100 consecutive mixed, and multiple series of negative and positive feedbacks.

Negative feedback (see Figures 22(a) and 22(d)). Negative feedback eliminates all possible worlds from the database that conflict with the feedback statement, i.e., that produce the associated query answer. Since feedback always targets a query answer that previously was given, negative feedback is sure to have some beneficial impact, because some uncertain data needs to be eliminated to prevent the answer to re-occur. It may happen, however, that only a local uncertainty is resolved such as whether to choose the TV guide's or IMDB's value of an actor's name. From the mostly gradual climb of both precision and recall, we can conclude that most of the possible negative feedback statements hit only local uncertainties and have some but little impact. Occasionally, however, a feedback statement has more impact (an upward jump in the graph). This occurs, for example, when feedback on an element's value (e.g., a certain actor name being incorrect) can only be true if the associated possible match between two movie data items is incorrect. Consequently, all data and uncertainty associated with this possible match of movies is eliminated having a significant impact on the quality of all queries.

We can furthermore observe that even with high impact negative feedback, the impact mainly affects precision while recall only marginally improves. This can be explained as follows. In terms of Figure 13, recall assesses the total probability mass of correct answers in the query result (expected value of $|C|$) relative to the number of correct answers ($|H|$). The latter clearly does not change as a result of feedback. Since we only remove incorrect data with negative feedback, the only effect of feedback on recall comes from slightly higher probabilities on some possibility nodes due to normalization, which produces a minor increase in the expected value of $|C|$. Precision assesses the ratio of probability mass of correct answers (expected value of $|C|$) in the complete query result (expected value of $|A|$). Since a high impact negative feedback statement may cause many other dependent incorrect answers to be eliminated, the total number of given answers is reduced significantly, hence the expected value of $|A|$ decreases. The latter effect causes precision to increase more than recall.

Positive feedback (see Figures 22(b) and 22(e)). Positive feedback show different behavior: it generally has more impact on recall than on precision. Moreover, in many cases it has no impact at all. A similar reasoning explains this behavior. Positive feedback eliminates all uncertain data conflicting with the feedback statement, i.e., all possible worlds that do not produce the corresponding query answer. If positive feedback concerns data coming from IMDB such as a keyword, this may completely resolve the entity match for the associated movie. Consequently all data associated with all incorrect matches for the movie is eliminated. In this case, the increase in probability mass for correct answers due to normalization (expected value of $|C|$) is substantial resulting in a substantial increase in recall. Resolving a movie entity ambiguity increases the probability mass for all data of that movie including the actor ambiguities 'inside' the movie.



(a) Precision (negative) (b) Precision (positive) (c) Precision (mixed)

(d) Recall (negative) (e) Recall (positive) (f) Recall (mixed)

(g) Size (h) Density (i) Decisiveness (j) Entropy

Fig. 22 Effects of user feedback (title margin 5; title threshold 0.4; actor threshold 0.4; precision and recall: average over all queries)

Therefore, for many queries the expected value of $|A|$ also increases resulting in a lesser rise in precision.

Sometimes a positive feedback statement has no impact at all. This happens when two alternative values for a certain answer are both correct. For example, ”The Godfather” and ”Godfather, The” are both considered as a correct answer for the title of this particular movie. Positive feedback on one eliminates the other, but the probability mass of both together doesn’t change, hence both $|A|$ and $|C|$ do not change.

The general trend of quality improvement is visible in all shorter series we ran, but the series differ in how fast quality improves as it depends on how ‘lucky’ our similated user is on stumbling on high impact feedback statements.

Mixed feedback (see Figures 22(c) and 22(f)). Mixed feedback achieves an improvement in precision from 0.915 to 0.968 and in recall from 0.921 to 0.962 with 100 feedback statements. Mixing positive and negative feedback succeeds in combining the strengths of both kinds of feedback, i.e., significant impact on both precision and recall. By combining negative and positive feedback, it takes a bit longer for precision to significantly improve than with negative feedback alone, but in this way both precision and recall jointly improve. Note that because our simulated user first picks a query and answer and then determines whether it should give negative or positive feedback on it, both kinds are unevenly distributed (75 positive and 25 negative).

Observe that no perfect quality is achieved. Analysis showed that the integrated data contains many small independent local uncertainties, for example, on the value for the name or role of an actor: ”Capt. Laurent” or ”Captain Laurent”. Since these are all independent, each needs a feedback

statement (negative of positive) to be resolved. Only 9 feedback statements pertained to actor names and roles, so with several thousand actors in our data set, only a small fraction is resolved. Furthermore, due to the limitations of our feedback algorithm, not all uncertain answers receive appropriate feedback, hence may remain unresolved.

Finally, although perhaps counter-intuitive, quality may go down as a result of feedback (see for example the temporary decline in precision and recall for one series in Figure 22(a) at 8 iterations). Suppose we have an imperfect match for a certain movie which results in four equally probable possibilities (see Figure 23, the leftmost one is the correct one). Movies are enriched with keywords from IMDB. It may happen that improperly matched movies carry some of the same keywords as the correct movie (suppose two incorrectly matched movies carry correct keyword A as well). A sketch of the situation is given in Figure 23(a). In the answer to query `//keyword`, keyword A receives a probability of $3 \times \frac{1}{4} = \frac{3}{4}$. Suppose that we then give feedback which results in the elimination of one of the incorrect matches (the rightmost one). After normalization, three possibilities remain with probability $\frac{1}{3}$ (see Figure 23(b)). If we re-evaluate the query, our correct keyword A now receives a probability of $2 \times \frac{1}{3} = \frac{2}{3} \leq \frac{3}{4}$, i.e., a reduction in probability. As a consequence expected recall and possibly also expected precision may decrease. What actually happened is that the query answer initially got a high probability *for the wrong reasons*, namely due to incorrectly matched movies that happened to have the same correct keyword. Our feedback did bring us closer to the truth and subsequent feedback iterations may remove all uncertainty, but due to 'less wrong reasons' the probability of a correct answer, hence overall quality, can go down temporarily.

Figures 22(g) through 22(j) show the effects of user feedback on the size and the amount of uncertainty. The size of the document gradually decreases as almost all feedback statements have the effect of removing subtrees somewhere in the document. As explained, low density and high decisiveness combined with low quality means that there is much locally certain data which is wrong, for example actor and location data from IMDB belonging to an incorrectly considered possible match for a movie. In our experiments, we see density increasing, decisiveness decreasing, while quality improves. We can therefore conclude that user feedback is effective in removing much of the locally certain but wrong data. The (local) uncertainty measures of density, decisiveness and entropy indicate a worsening of the uncertainty at first. If perfect quality is achieved, we know that density is 0, decisiveness is 1, and entropy is 0. Therefore, at some point density and entropy must come down, and decisiveness must go up. Although we do not achieve perfect quality in our experiments, an improvement of the uncertainty measures can be observed from about 70 iterations. It would be interesting to investigate whether there exist adapted uncertainty metrics that better assess the global situation of the uncertainty.

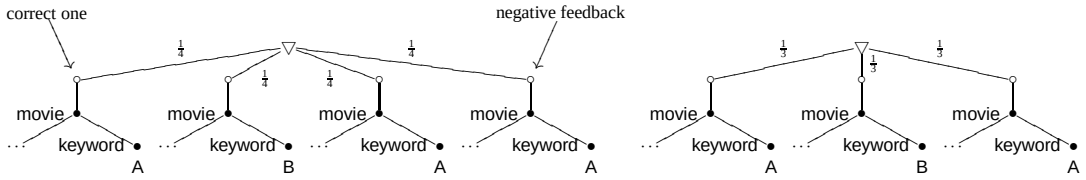
Evidence for effectiveness of user feedback. The first observation that we like to emphasize is that negative feedback is capable of consistently improving the quality of the integrated data set. By mixing positive with negative feedback, in most cases precision and recall jointly improve. Note that we started with the integration result under the worst uncertainty conditions (worst margin and thresholds).

In general, jumps in quality improvement are possible when there exist dependencies between probabilistic events in the data (in our case, actor entity resolution possibilities and enriched data from IMDB depend on movie entity resolution possibilities). For our application and data context, we observe numerous high impact jumps in each series where our simulated random user stumbled upon a high impact feedback statement. There are ways imaginable to establish beforehand for which values in the query answer positive or negative feedback would have big impact on the integration quality. By bringing those answers to the user's attention, i.e., asking for targeted feedback, we may obtain much more rapid quality improvements. Finally, the same call for further experiments and carefulness in drawing conclusions as mentioned at the end of the previous section apply to the experimental set-up of this section.

7 Conclusions and Future Work

Entity resolution is a challenging problem in many applications that require different data sources to be integrated, such as portals. An often-used rule of thumb states that about 90% of the development effort is devoted to solving the remaining 10% hard cases. Our probabilistic integration approach aims at reducing the development effort needed for such applications by allowing some semantic uncertainty to remain in the data, while still being able to meaningfully use this data. The developer is only required to provide a few knowledge rules and rough estimations for thresholds to produce a usable *initial* integration.

The main contribution of this paper is a thorough experimental investigation of the effects and sensitivity of rule definition, threshold tuning, and user feedback on the integration quality. Our experiments are based on an application for which movie data from a TV guide is enriched with data from IMDB. Essential to the enrichment is resolution of the entities movie and actor that occur in both data sources. We first showed that for the enrichment of the top-100 movies from the TV movie guide and a realistic and substantial data set of 243,856 movies from IMDB, indeed 85% to 92% of the cases were easy to resolve with a few simple rules and



(a) Example sketch before feedback (b) Example sketch after feedback

Fig. 23 Example sketch where quality initially goes down after user feedback

rough thresholds. We then focused on investigating the *qualitative* effects of varying thresholds and rule definitions. By measuring the expected value of average precision and recall for 43 queries, we determined the quality of the integrated data for various threshold settings and rule configurations. We also showed that exponential data growth does not occur even under challenging conditions, which is mainly due to the proposed clustering phase of the integration algorithm. The results provide convincing evidence that our approach indeed reduces development effort — and not merely shifts the effort to rule definition and threshold tuning — by showing that setting rough safe thresholds and defining only a few rules suffices to produce a 'good enough' integration that can be meaningfully used.

We like to emphasize that this pertains to the quality of the *initial* data integration. In our probabilistic integration approach, the knowledge rules are only needed to quickly produce an integrated data set of manageable size that can be meaningfully used by an application. We took an integrated data set with a lot of uncertainty and subjected it to series of positive, negative, and mixed user feedback to show that feedback obtained from a user interacting with an application is effective in quickly improving the quality of the integrated data, i.e., in resolving the 10% hard cases the developer didn't try to solve. Even with a limited user feedback algorithm, the experiments already confirm this. Moreover, it appears that mixing negative and positive feedback is important to improve both precision and recall. A beforehand analysis of feedback on which query answers will have high impact seems possible and may accelerate quality improvement even more.

Several aspects of our approach call for further research. We plan to continue investigating the interplay between (formulations of) knowledge rules, properties of the integrated data, and quality of query answers, e.g., in other real-life application domains. More insight into these aspects is expected to be important for devising effective tools to support developers in using our approach in practice. For the latter, further research is also needed on how to seamlessly embed feedback giving functionality in GUIs, and on improving the scalability of the algorithms for integration, querying and user feedback in the underlying probabilistic database. We also need to compare our new quality measure, expected precision and recall, with other quality measures to show its strength. It turned out that new measures capable of giving a more global indication of the degree of uncertainty are highly desirable. Research is also needed to investigate the possible correlation between the uncertainty measures and quality; this would have much impact because in that case the uncertainty measures can help with threshold tuning.

Feedback at present is merciless, i.e., if incorrect feedback is given, correct information may get lost. We like to investigate ways to benefit from opinions and other feedback that itself can not be fully trusted. Furthermore, our integration approach is expected to benefit from being combined with fuzzy querying techniques. Combining our approach with schema matching has the potential of further decreasing development effort for data integration. Finally, we believe that our probabilistic integration approach can be adapted to a (probabilistic) *relational* database setting.

References

[AKO07] L. Antova, C. Koch, and D. Olteanu. MayBMS: Managing incomplete information with probabilistic world-set decompositions. In *Proc. of the 23nd Int'l Conf. on Data Engineering (ICDE), Istanbul, Turkey*, pages 1479–1480, April 2007.

[AS06] S. Abiteboul and P. Senellart. Querying and updating probabilistic information in XML. In *Proc. of the Int'l Conf. on Extending Database Technology (EDBT), Munich, Germany*, pages 1059–1068, 2006. LNCS 3896.

[BDM+05] J. Boulos, N.N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *Proc. of the 2005 ACM SIGMOD Int'l Conf. on Management of data, Baltimore, Maryland, USA*, pages 891–893, 2005.

[BGMM+09] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *VLDB Journal*, 2009.

[BGMP90] D. Barbará, H. Garcia-Molina, and D. Porter. A probabilistic relational data model. In *Proc. of the Int'l Conf. on Extending Database Technology (EDBT) Venice, Italy*, volume 416 of *LNCS*, pages 60–74. Springer, March 1990. ISBN 3-540-52291-3.

[BGvK+06] P.A. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner. MonetDB/XQuery: a fast XQuery processor powered by a relational engine. In *Proc. of SIGMOD, Chicago, IL*, pages 479–490, 2006.

[BSHW06] O. Benjelloun, A. Das Sarma, C. Hayworth, and J. Widom. An introduction to ULDBs and the Trio system. *IEEE Data Engineering Bulletin*, 29(1):5–16, 2006.

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999. ISBN 0-201-39829-X.

[CCX08] R. Cheng, J. Chen, and X. Xie. Cleaning uncertain data with quality guarantees. In *Proc. of the 34th Int'l Conf. on Very Large Data Bases (VLDB), Auckland, New Zealand*, pages 722–735, August 2008.

[CSP05] R. Cheng, S. Singh, and S. Prabhakar. U-DBMS: A database system for managing constantly-evolving data. In *Proc. of the 31st Int'l Conf. on Very Large Data Bases (VLDB), Trondheim, Norway*, pages 1271–1274, 2005.

[CYC07] T. Cheng, X. Yan, and K. Chen-Chuan Chang. EntityRank: Searching entities directly and holistically. In *Proc. of the 33rd Int'l Conf. on Very Large Data Bases (VLDB), Vienna, Austria, September 23-27, 2007*, pages 387–398. ACM, 2007.

[DDH01] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data, Santa Barbara, California, USA*, pages 509–520, May 2001. ISBN 1-58113-332-4.

[DDH03] A. Doan, P. Domingos, and A.Y. Halevy. Learning to match the schemas of data sources: A multistrategy approach. *Machine Learning*, 50(3):279–301, 2003.

[DH05] A. Doan and A.Y. Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine*, 2005.

[DHY07] X. Luna Dong, A.Y. Halevy, and C. Yu. Data integration with uncertainty. In *Proc. of the 33rd Int'l Conf. on Very Large Data Bases (VLDB), Vienna, Austria, September 23-27*, pages 687–698. ACM, 2007.

[dKvK07a] A. de Keijzer and M. van Keulen. Quality measures in uncertain data management. In *Proc. of the 1st Int'l Conf. on Scalable Uncertainty Management (SUM), Washington, DC*, volume 4772 of *LNCS*, pages 104–115, 2007.

[dKvK07b] A. de Keijzer and M. van Keulen. User feedback in probabilistic integration. In *2nd Int'l Workshop on Flexible Database and Information System Technology (FlexDBIST), Regensburg, Germany*, pages 377–381, Los Alamitos, September 2007.

[dKvK08] A. de Keijzer and M. van Keulen. IMPrECISE: Good-is-good-enough data integration. In *Proc. of the 24th Int'l Conf. on Data Engineering (ICDE), Cancun, Mexico*, April 2008.

[dKvKL06] A. de Keijzer, M. van Keulen, and Y. Li. Taming data explosion in probabilistic information integration. In *Online Pre-Proc. of IIDB, Munich, Germany*, pages 82–86, 2006. Position paper. `http://ssi.umh.ac.be/iidb`.

[dR95] M. de Rougemont. The reliability of queries. In *Proc. of the 14th ACM Symposium on Principles of Database Systems (PODS), San Jose, CA*, pages 286–291, May 1995.

[DS+07] P. DeRose, W. Shen, F. Chen 0002, Y. Lee, D. Burdick, A. Doan, and R. Ramakrishnan. Dblife: A community information management platform for the database research community (demo). In *Proc. of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, CA*, pages 169–172, January 2007.

[DSC+07] P. DeRose, W. Shen, F. Chen, A. Doan, and R. Ramakrishnan. Building structured web community portals: A top-down, compositional, and incremental approach. In *Proc. of the 33rd Int'l Conf. on Very Large Data Bases (VLDB), Vienna, Austria*, pages 399–410. ACM, September 2007.

[FGM07] F. Furfaro, S. Greco, and C. Molinaro. A three-valued semantics for querying and repairing inconsistent databases. *Annals of Mathematics and Artificial Intelligence*, 51(2-4):167–193, 2007.

[Gal08] A. Gal. Interpreting similarity measures: Bridging the gap between schema matching and data integration. In *Proc. of the Workshop on Information Integration Methods, Architectures, and Systems (IIMAS), Cancun, Mexico*, pages 278–285, April 2008.

[GGH98] E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *Proc. of the 17th ACM Symposium on Principles of Database Systems (PODS), Seattle, WA*, pages 227–234, June 1998.

[HGS03] E. Hung, L. Getoor, and V.S. Subrahmanian. PXML: A probabilistic semistructured data model and algebra. In *Proc. of the 19th Int'l Conf. on Data Engineering (ICDE), Bangalore, India*, page 467, March 2003. ISBN 0-7803-7665-X.

[HL06] A. Hunter and W. Liu. Merging uncertain information with semantic heterogeneity in XML. *Knowledge and Information Systems*, 9(2):230–258, February 2006.

[KD08] B. Kanagal and A. Deshpande. Online filtering, smoothing and probabilistic modeling of streaming data. In *Proc. of the 24th Int'l Conf. on Data Engineering (ICDE), Cancun, Mexico*, pages 1160–1169, April 2008.

[KKA05] M. van Keulen, A. de Keijzer, and W. Alink. A probabilistic XML approach to data integration. In *Proc. of the 21st Int'l Conf. on Data Engineering (ICDE), Tokyo, Japan*, pages 459–470, April 2005.

[KO08] C. Koch and D. Olteanu. Conditioning probabilistic databases. In *Proc. of the 34th Int'l Conf. on Very Large Data Bases (VLDB2008), Auckland, New Zealand*, pages 313–326, August 2008.

[LLRS97] L.V.S. Lakshmanan, N. Leone, R. Ross, and V.S. Subrahmanian. ProbView: a flexible probabilistic database system. *ACM Transactions on Database Systems (TODS)*, 22(3):419–469, 1997.

[MBGM06] D. Menestrina, O. Benjelloun, and H. Garcia-Molina. Generic entity resolution with data confidences. In *Proc. of the First Int'l VLDB Workshop on Clean Databases (CleanDB), Seoul, Korea*, September 2006.

[MM07] M. Magnani and D. Montesi. Uncertainty in data integration: current approaches and open problems. In *Proc. of the 1st Int'l Workshop on Management of Uncertain Data (MUD), Vienna, Austria*, number WP07-08 in CTIT Workshop Proceedings, September 2007. ISSN 0929-0672.

[MSC06] D. Milano, M. Scannapieco, and T. Catarci. Structure aware xml object identification. In *Proc. of the 1st Int'l VLDB Workshop on Clean Databases (CleanDB), Seoul, Korea*, September 2006.

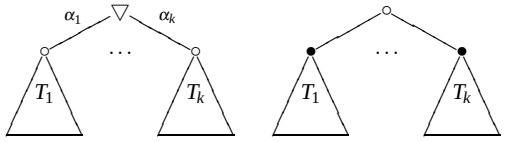
[Orr98] Ken Orr. Data quality and systems theory. *Communications of the ACM*, 41(2):66–71, 1998.

[SDH08] A. Das Sarma, X. Dong, and A. Halevy. Bootstrapping pay-as-you-go data integration systems. In *Proc. of the 2008 ACM SIGMOD Int'l Conf. on Management of data (SIGMOD), Vancouver, Canada*, pages 861–874, 2008.

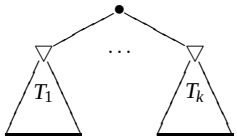
[SH08] P. Serdyukov and D. Hiemstra. Modeling documents as mixtures of persons for expert finding. In *Proc. of the 30th European Conf. on IR Research (ECIR), Glasgow, UK*, volume 4956 of *LNCS*, pages 309–320. Springer, April 2008.

[vK08] R. van Kessel. Querying probabilistic xml. Master's thesis, Univ. of Twente, Enschede, Netherlands, April 2008.

[Wij06] J. Wijsen. Project-join-repair: An approach to consistent query answering under functional dependencies. In *Proc. of the 7th Int'l Conf. on Flexible Query Answering Systems (FQAS), Milan, Italy*, volume 4027 of *LNCS*, pages 1–12. Springer, June 2006.



(a) Case 1: probability node (b) Case 2: possibility node



(c) Case 3: XML node

Fig. 24 Three cases in recursive descent

A Proof of correctness entropy calculation

Let n be the number of possible worlds and p_j be the probability of world j, then $\mathsf{E} = -\sum_{j=1}^{n} p_j \log_2 p_j$.

We calculate the entropy in a recursive descent fashion analogous to the construction of the set of all possible worlds. The algorithm for the constructing the set of possible worlds is as follows[KKA05]: at the current node we construct for each child the set of possible subtrees of that child. Let p_i^j be the probability of possible subtree j of child i. There are three cases: our current node is either a probability node, a possibility node, or an XML node (see Figure 24).

- In case 1, we take the union of all possible subtrees, i.e., $\bigcup_{i=1\ldots n} T_i$. The probability of each subtree becomes $\alpha_i p_i^j$.
- In cases 2 and 3, we take all combinations of selecting one possible subtree j_i from each child i. The probability of each combination is $\prod_{i=1}^{k} p_i^{j_i}$.

As the subtree of each node represents the set of possible subtrees of that node, we can define the entropy of a node to be the entropy of the set of possible subtrees. Our base case is an XML leaf node which represents the set of one possible subtree with probability 1. The entropy of this set is $\mathsf{E} = 1\log_2 1 = 0$.

For subtrees T_i $(i = 1\ldots k)$, let E_i be the entropy of the set of possible subtrees of T_i, n_i the number of possible subtrees of T_i, and p_j^i the probability of possible subtree j of T_i. Since $\sum_{j=1}^{n_i} p_j = 1$ we define $\tilde{p}_i = \sum_{j=1}^{n_i-1} p_j^i$. Hence

$$\begin{aligned}\mathsf{E}_i &= -\sum_{j=1}^{n_i} p_j^i \log_2 p_j^i \\ &= -\sum_{j=1}^{n_i-1} p_j^i \log_2 p_j^i + (1-\tilde{p}_i)\log_2(1-\tilde{p}_i) \qquad (1)\end{aligned}$$

A.1 To be proven

We claim that

- Case 1: $\mathsf{E} = -\sum_{i=1}^{k} \alpha_i \log_2 \alpha_i - \alpha_i \mathsf{E}_i$
- Cases 2 and 3: $\mathsf{E} = -\sum_{i=1}^{k} \mathsf{E}_i$

A.2 Case 1

Since we take the union of possible substrees and each probability is multiplied by α_i, the entropy can be calculated as follows.

$$\mathsf{E} = -\sum_{i=1}^{k}\sum_{j=1}^{n_i} \alpha_i p_j^i \log_2 \alpha_i p_j^i$$

By Equation 1:

$$= -\sum_{i=1}^{k}\left(\sum_{j=1}^{n_i-1} \alpha_i p_j^i \log_2 \alpha_i p_j^i + \alpha_i(1-\tilde{p}_i)\log_2 \alpha_i(1-\tilde{p}_i)\right)$$

Since $\log_2 xy = \log_2 x + \log_2 y$:

$$= -\sum_{i=1}^{k}\left(\begin{array}{l}\sum_{j=1}^{n_i-1} \alpha_i p_j^i \log_2 \alpha_i + \alpha_i p_j^i \log_2 p_j^i \\ \qquad + \alpha_i(1-\tilde{p}_i)\log_2 \alpha_i + \alpha_i(1-\tilde{p}_i)\log_2(1-\tilde{p}_i)\end{array}\right)$$

Simple rewriting

$$= -\sum_{i=1}^{k}\left(\begin{array}{l}\alpha_i \log_2 \alpha_i \left(\sum_{j=1}^{n_i-1} p_j^i + 1 - \tilde{p}_i\right) \\ \quad + \alpha_i \left(\sum_{j=1}^{n_i-1} p_j^i \log_2 p_j^i + (1-\tilde{p}_i)\log_2(1-\tilde{p}_i)\right)\end{array}\right)$$

By definition of $\tilde{p}_i$ and E_i:

$$= -\sum_{i=1}^{k} \alpha_i \log_2 \alpha_i - \alpha_i \mathsf{E}_i$$

A.3 Cases 2 and 3

For these two cases, we take all combinations of selecting one possible subtree j_i from each child i where the probability of each combination is $\prod_{i=1}^{k} p_i^{j_i}$. Therefore, the entropy can be calculated as follows.

$$\mathsf{E} = -\sum_{j_1=1}^{n_1}\cdots\sum_{j_k=1}^{n_k} p_{j_1}^1 \cdots p_{j_k}^k \log_2 p_{j_1}^1 \cdots p_{j_k}^k$$

By $\log_2 xy = \log_2 x + \log_2 y$:

$$= -\sum_{j_1=1}^{n_1}\cdots\sum_{j_k=1}^{n_k} p_{j_1}^1 \cdots p_{j_k}^k (\log_2 p_{j_1}^1 + \cdots + \log_2 p_{j_k}^k)$$

We rewrite by grouping all terms with the same log argument:

$$= -\left(\begin{array}{l}\sum_{j_1=1}^{n_1} p_{j_1}^1 \log_2 p_{j_1}^1 \left(\sum_{j_2=1}^{n_2}\cdots\sum_{j_k=1}^{n_k} p_{j_2}^2 \cdots p_{j_k}^k\right) \\ \qquad + \\ \qquad \vdots \\ \qquad + \\ \sum_{j_k=1}^{n_k} p_{j_k}^1 \log_2 p_{j_k}^k \left(\sum_{j_1=1}^{n_1}\cdots\sum_{j_{k-1}=1}^{n_{k-1}} p_{j_1}^1 \cdots p_{j_{k-1}}^{k-1}\right)\end{array}\right)$$

Since for each T_i we consider *all* possible subtrees, we know that their probabilities add up to one, i.e., $\sum_{j_i=1}^{n_i} p_{j_i}^i = 1$. Therefore,

$$= -\left(\sum_{j_1=1}^{n_1} p_{j_1}^1 \log_2 p_{j_1}^1 + \cdots + \sum_{j_k=1}^{n_k} p_{j_k}^1 \log_2 p_{j_k}^k\right)$$

By definition of E_i:

$$= -\sum_{i=1}^{k} \mathsf{E}_i$$